**COMPILER DESIGN PROJECT DOCUMENTATION**

**Lexical and Semantic Analysis of Descriptive answers in Exams to understand Student knowledge**

|  |
| --- |
| **DATE: 06/11/2020**  **BATCH : N**  **TEAM : 5**  **SUBJECT CODE : CS6109**  **SUBJECT NAME : COMPILER DESIGN** |

**Submitted by,**

**NIDYA M 2018103046**

**NIRANJANA S 2018103570**

**SARIKAA S 2018103058**

**Table of contents:**

* Abstract
* Introduction
* Objective
* Literature Survey
* Architecture diagram
* Modules
  + Module Explanation
  + Pseudocode
* Output
* Conclusion
* Reference

**Lexical and Semantic Analysis of Descriptive answers in Exams to understand Student knowledge**

**Abstract :**

Students’ writing in exams can provide better understanding into their thinking than multiple-choice questions. However, time and resource constraints often prevent teachers from using writing assessments in most of the courses. We looked over the use of computer software to analyze student writing and to understand their conceptual knowledge. These analyses help to see the conceptual understanding of the students and their difficulties explaining these topics and their ideas. This suggests that computerized lexical analysis may be useful for automatically categorizing large numbers of student descriptive answers. Lexical analysis provides teachers better insight into student thinking and a whole-class perspective that are difficult to obtain from multiple-choice questions or reading individual responses.

**Terms used:**

Lexical analysis , expert answer , conceptual understanding and barriers.

**Objective of the project :**

Effective Assessment is a key component of engaging student learning with understanding. However, detecting the conceptual understanding of students with multiple choice questions is often difficult, because they do not reveal students’ reasoning. One way we choose to investigate student understanding and reasoning is by using constructed response or essay questions. However, the large enrollment numbers make the evaluation of descriptive questions difficult. It is sometimes suggested that it may not provide an accurate depiction of the whole class, as a few exceptionally good or poor responses may alter the teachers’ perception of the class’s understanding. Rapid reading and grading of these responses by the teacher may also lead to errors in scoring or missing important conceptual connections made by students. To resolve these problems, we are using lexical analysis software to analyze and evaluate student descriptive answers to provide rapid feedback for the teacher.

**Project Introduction:**

The project is about automatic correction of descriptive answers and analysing the students' understanding. The steps included are:

* Checking spelling and ignore it
* Checking grammar and ignore it
* Parsing the sentence construct
* Based on the result, evaluating the marks and assessing the student’s understanding

The language we have chosen for this project is python3. So far , we have completed 2 modules.

**Literature Survey**

Evaluation of subjective or text based answers has been a hurdle in the development of eLearning systems since a very long time. The problem related to evaluation of subjective answers is that each student has his/her own way of answering and it is difficult to determine the degree of correctness.Typically, there are two types of question answering systems:

Closed-domain question answering

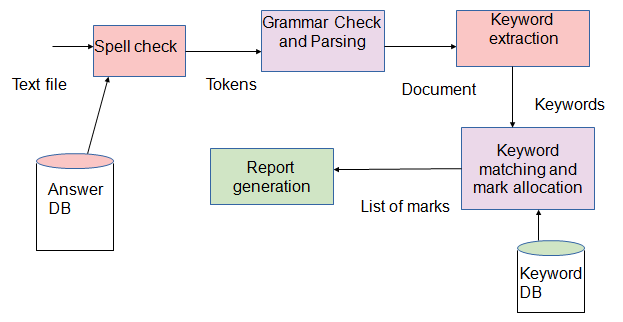
1. This deals with questions under a specific domain.
2. It is harder on the other side since information is not generally available in the public domain.

Open-domain question answering

1. This deals with questions about nearly everything.
2. These rely only on general ontology and world knowledge

Many architectures and features have been proposed for descriptive answer evaluation. The approaches are mainly based on keyword matching, sequence matching and quantitative analysis, but semantic analysis of descriptive answers is still an open problem. Considering the general structure of text analysis in natural language processing, most of the work has been done for morphological and syntactic analysis, but semantic, pragmatic and discourse are still being explored. Online tools that support managing online assessments such as Moodle and Zoho are based on string matching techniques for short answers but long answer evaluation is still handled manually by most systems. Features which are available currently in online assessment are: Question paper setting Online Evaluation of objective type questions Question bank editor Spell checker Grammar checker Report generation of result Descriptive answer evaluation is still an open problem.

**Architecture Diagram**

****

**List Of Modules**

1. Module 1 - Spell check
2. Module 2 - Grammar check
3. Module 3 - Keyword extraction
4. Module 4 - Keyword matching and mark allocation
5. Module 5 - Total mark and remarks

**Modules Description**

**Module 1 : Spell check**

In module 1, we have implemented lexical analysis of the text document. The steps are as follows:

* Saving the answers as the text document
* Giving it as an input
* Used textblob library of python
* Saving the list of terms that are frequently used in the subject as dictionary for reference
* Matching the replacements suggestions with the user defined dictionary and replacing the wrongly spelled word with the suitable word
* The reason for doing the spelling check is not to reduce marks for the spelling errors

**Pseudocode**

# import TextBlob

from textblob import TextBlob

fru= TextBlob("Appl nd orange arr fruuts.")

# using TextBlob.correct() method

fru= fru.correct()

print(fru)

**Module 2 : Grammar check**

In module 2, we have implemented syntactic analysis of the text document. We are verifying whether the text document contains any grammatical error. If grammatical error is detected it will correct it so that there won't be any reduction of marks for grammatical errors. The steps are as follows:

* Output from the module 1 as used as input
* Using python language-check library
* Comparing the tokens with the dictionary

**Pseudocode**

>>> import language\_check

>>> tool = language\_check.LanguageTool('en-US')

>>> text = u'Most ofthe childrns like the aplles'

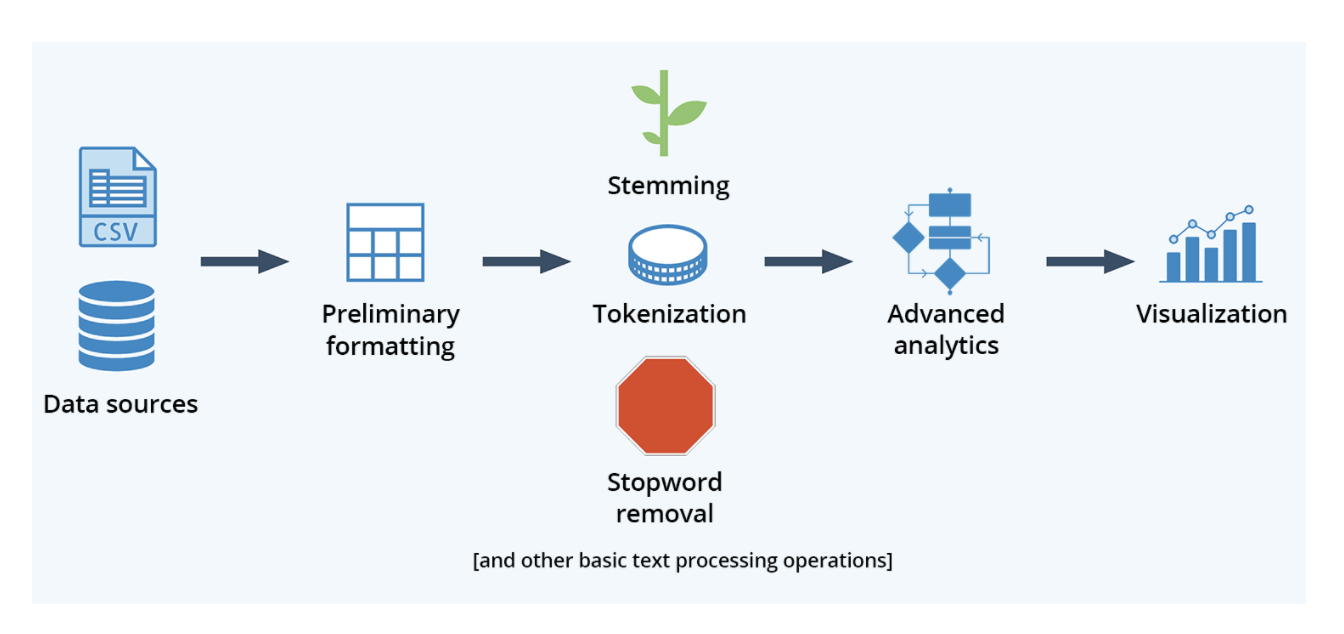
>>> matches = tool.check(text)

>>> len(matches)

**Module 3 : Keyword extraction**

In module 3, to extract the keywords from the file, we are using RAKE’s algorithm. The rake’s algorithm works as follows:

* Split the text document into words.
* Declare the list of stopwords and delimiters.
* Remove the stopwords and delimiters from the splitted words.
* Then it will set degree score to the words according to the frequency and the occurrence along with the other words
* Based on this, if two words occur together frequently, it will tag it as a single keyword

****

**Pseudocode for Rake algorithm :**

**RAKE Algorithm:**

build\_stop\_word\_regex(stop\_word\_list):

stop\_word\_regex\_list = []

for word in stop\_word\_list:

word\_regex = '\\b' + word + '\\b'

stop\_word\_regex\_list.append(word\_regex)

stop\_word\_pattern = re.compile('|'.join(stop\_word\_regex\_list), re.IGNORECASE)

return stop\_word\_pattern

adjoined\_candidates\_from\_sentence(s, stoplist, min\_keywords, max\_keywords):

for num\_keywords in range(min\_keywords, max\_keywords + 1):

for i in range(0, len(sl) - num\_keywords):

if sl[i] not in stoplist:

candidate = sl[i]

j = 1

keyword\_counter = 1

contains\_stopword = False

while keyword\_counter < num\_keywords and i + j < len(sl):

candidate = candidate + ' ' + sl[i + j]

if sl[i + j] not in stoplist:

keyword\_counter += 1

else:

contains\_stopword = True

j += 1

if contains\_stopword and candidate.split()[-1] not in stoplist and keyword\_counter == num\_keywords:

candidates.append(candidate)

return candidates

calculate\_word\_scores(phraseList):

word\_frequency = {}

word\_degree = {}

for phrase in phraseList:

word\_list = separate\_words(phrase, 0)

word\_list\_length = len(word\_list)

word\_list\_degree = word\_list\_length - 1

for word in word\_list:

word\_frequency.setdefault(word, 0)

word\_frequency[word] += 1

word\_degree.setdefault(word, 0)

word\_degree[word] += word\_list\_degree # orig.

for item in word\_frequency:

word\_degree[item] = word\_degree[item] + word\_frequency[item]

word\_score = {}

for item in word\_frequency:

word\_score.setdefault(item, 0)

word\_score[item] = word\_degree[item] / (word\_frequency[item] \* 1.0)

return word\_score

**MODULE 4 : Keyword matching and mark allocation**

In this module, extracted keywords are matched with the user defined keywords and the marks are allocated according to the length, errors and grammar check. The final answer for each question is stored in the array. The array is updated each time the answer is submitted before finishing the test. The steps are as follows:

* Creating GUI Interface using tkinter library.
* Displaying questions from text file “questions.txt”.
* Initialisation of total mark array as zero.
* nex()-moves to next question
* prev()-moves to previous question
* enfunc()-Mark calculation at each stage.
* Total mark array is updated after each submission.

**Pseudocode:**

totmark[5] = {0,0,0,0,0,0}

tkinkter geometry(){

// how the gui is going to be formatted

}

nex(){

// moves to the next function

if(counter>6)

Print “limit exceeded”

}

prev(){

// moves to the previous function

if(counter==0)

Print “this is the first question”

}

enfunc(){

rake(){

//rake algorithm in module 3

}

//spell che

ck form module 1

// grammar check from module 2

count = count + 1 //for errors

// allocate marks as per errors

c = marks3

// count length

len = len +1;

// marks allocation for len

a = marks1

for key in line

for key1 in file

if key == key1

marks2 = mark2 +1

b = marks2

**MODULE 5 : Total mark and remarks**

In this module, the total marks for the test will be displayed after clicking the finish the button of the test window. By clicking the show report button, the analysis report is generated depicting the bar graph with marks along the x axis and question numbers along the y axis.The steps are as follows:

* finish()-total mark calculation and display
* showrpt()-shows bar graph for the marks generated using matplotlib library.
* X-axis - marks out of 10
* Y-axis - question numbers
* After clicking the exit button on the tkinter window the tkinter window closes.

**Pseudocode**

finish(){

total = a+b+c

totmark[counter-1] = total

}

showrpt(){

//bar graph using matplotlib

for i in totmark

if 0<=i<=3

Print “not known”

If 4<=i<=5

Print “vaguely understood”

If 6<=i<=7

Print “moderate”

If 8<=i<=10

Print “satisfactory”

}

**Result Discussion and Summary:**

Apart from printing the marks evaluated for the test, this program helps to visualize the conceptual understanding of the students, via bar graph.

The rake algorithm works properly only if the answer includes any of the predefined smartstop lists other than the whitespace. Thus the answer has to be in a phrase or sentence format. Otherwise, it allocates marks only for the length of the answer even if the required keywords are detected. This is the advantage of this algorithm. We implemented this algorithm in order to prevent malpractices like, typing only keywords to manipulate the system.

**Module 1 and 2:**



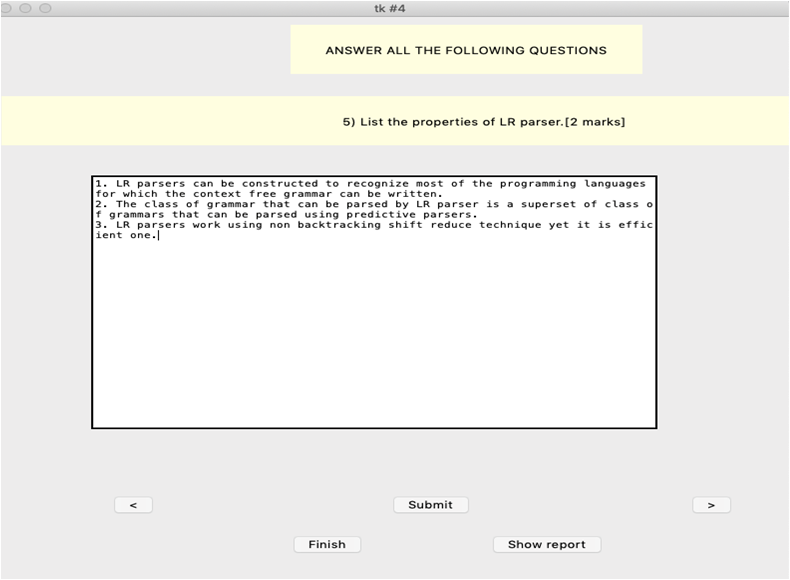


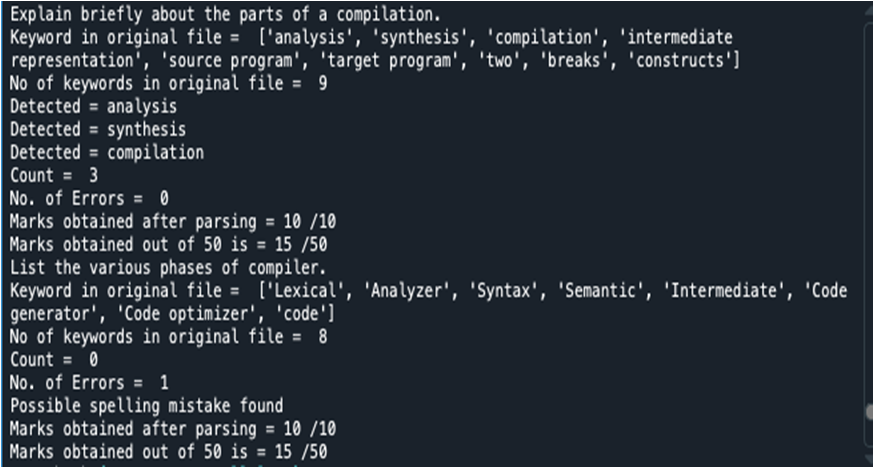
**Module 3:**

****

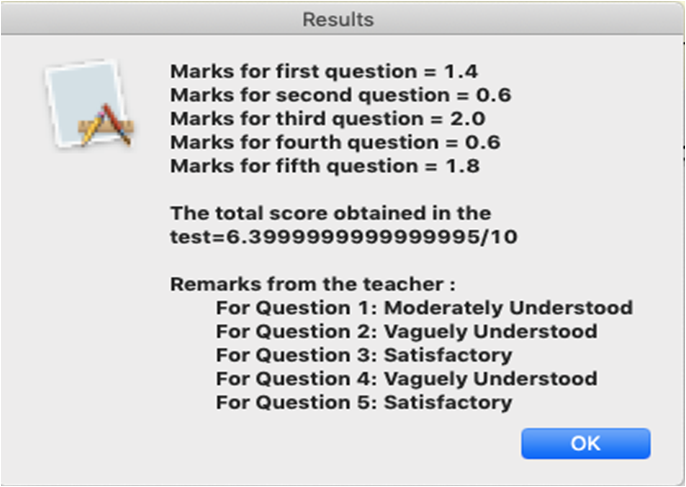
****

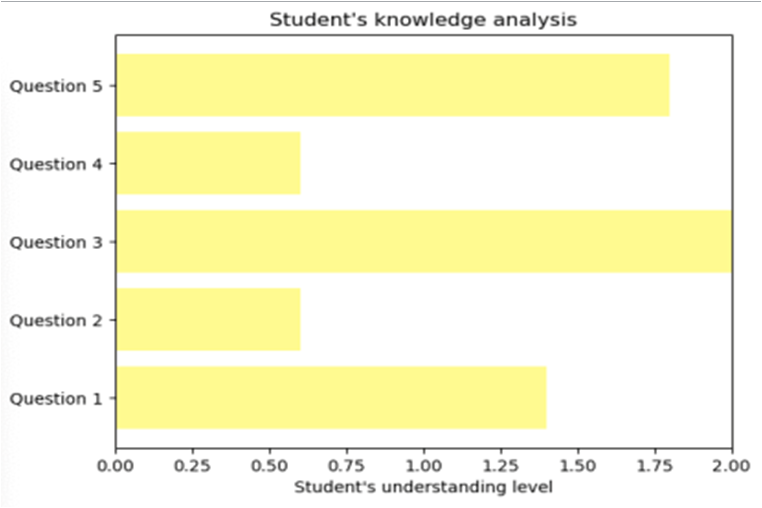
**Module 4:**

****

****

**Module 5:**

****

****

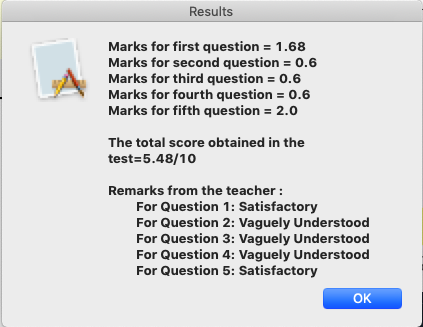
**Tabular column of INPUT & OUTPUT**

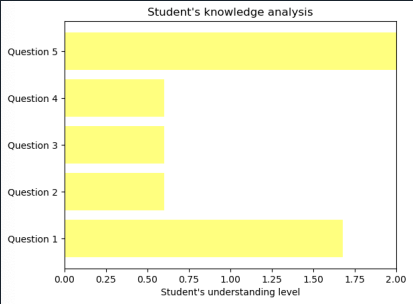
Comparison of answers among three different students

**Student 1**

|  |  |  |
| --- | --- | --- |
| **Question no** | **Input** | **Output** |
| 1 | A compiler is a special program that processes statements written in a particular programming language and turns them into machine language | Count = 6  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 42 /50 |
| 2 | There are two parts to compilation: analysis and synthesis. | Count = 3  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 15 /50 |
| 3 | Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variable | Count = 2  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 15 /50 |
| 4 | We basically have two phases of compilers, namely Analysis phase and Synthesis phase. | Count = 0  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 15 /50 |
| 5 | The LR parser is a non recursive, shift reduce, bottom up parser. It uses a wide class of context-free grammar | Count = 11  No. of Errors = 2  Possible spelling mistake found  Possible spelling mistake found  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 50 /50 |

**Overall Result :**

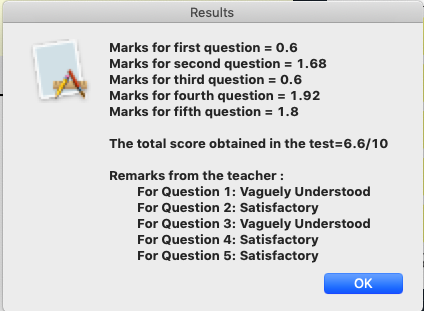
****

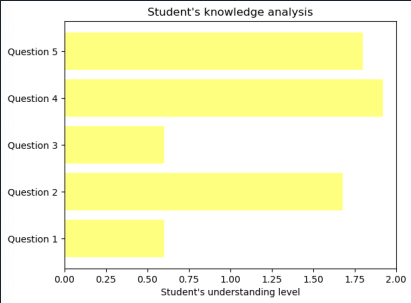
****

**Student 2**

|  |  |  |
| --- | --- | --- |
| **Question No** | **Input** | **Output** |
| 1 | A program that converts instructions into a machine code or lower level form so that they can be read and executed by a computer. | Count = 3  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 15 /50 |
| 2 | The analysis part breaks up the source program into constituent pieces. The synthesis part constructs the target program from the intermediate code. | Count = 6  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 42 /50 |
| 3 | Symbol table is used to store the information about the occurrence of various entities. | Count = 0  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 15 /50 |
| 4 | Lexical Analysis  Syntax analysis or parsing  Semantic Analysis  Intermediate Code Generation  Code Optimization  Code Generation | Count = 7  No. of Errors = 1  You used three nouns after one another, and this might decrease readability. You might consider rewording the sentence.  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 48 /50 |
| 5 | Works on smallest class of grammar  Few number of states  Simple and fast construction | Count = 7  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 45 /50 |

**Overall Result :**

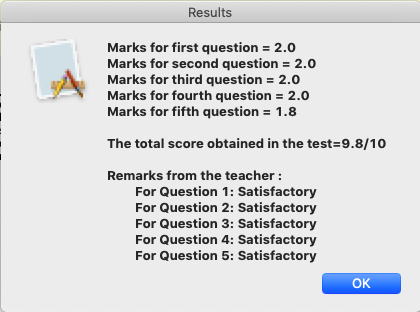
****

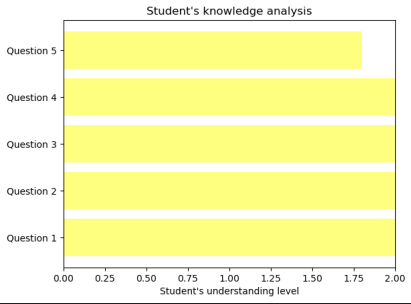
****

**Student 3**

|  |  |  |
| --- | --- | --- |
| **Question No** | **Input** | **Output** |
| 1 | A compiler is a program that reads a program written in one language –the source language and translates it into an equivalent program in another language-the target language. The compiler reports to its user the presence of errors in the source program. | Count = 14  No. of Errors = 2  Possible spelling mistake found  Possible spelling mistake found  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 50 /50 |
| 2 | Analysis and Synthesis are the two parts of compilation. The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program. The synthesis part constructs the desired target program from the intermediate representation. | Count = 10  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 50 /50 |
| 3 | A symbol table is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrieve data from that record quickly. | Count = 7  No. of Errors = 0  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 50 /50 |
| 4 | The following are the various phases of a compiler:  Lexical Analyzer  Syntax Analyzer  Semantic Analyzer  Intermediate code generator  Code optimizer  Code generator | Count = 13  No. of Errors = 1  You used three nouns after one another, and this might decrease readability. You might consider rewording the sentence.  Marks obtained after parsing = 10 /10  Marks obtained out of 50 is = 50 /50 |
| 5 | 1. LR parsers can be constructed to recognize most of the programming languages for which the context free grammar can be written.  2. The class of grammar that can be parsed by LR parser is a superset of class of grammars that can be parsed using predictive parsers.  3. LR parsers work using non backtracking shift reduce technique yet it is an efficient one. | Count = 27  No. of Errors = 3  Possible spelling mistake found  Possible spelling mistake found  Possible spelling mistake found  Marks obtained after parsing = 5 /10  Marks obtained out of 50 is = 45 /50 |

**Overall Result :**

****

****

**Conclusion :**

The descriptive assessment gives a good understanding of how the students think than multiple choice questions.The categories and the custom libraries can be used to visualize the trends in the students’ writing and their conceptual barriers.The teachers can immediately train the students accordingly to make them understand the concepts well.Thus this system will help in improving the students’ concepts and understanding.

**References :**

1. https://monkeylearn.com/keyword-extraction/Rake’s algorithm
2. Hanxiao Shi, Guodong Zhou and Peide Qian (2010), ”An Attribute based Sentiment Analysis System”
3. Papri Chakraborty (2012), ”Developing an Intelligent Tutoring System for Assessing Students’ Cognition and Evaluating Descriptive Type Answer”
4. Mita K. Dalal, Mukesh A. Zave (2011), “Automatic Text Classification: A Technical Review”, International Journal of Computer Applications
5. https://pypi.org/project/language-check/
6. Implementation of RAKE - Rapid Automatic Keyword Extraction algorithm Rose, S., D. Engel, N. Cramer, and W. Cowley (2010).