**Part-II, 3rd Elective**
**COMPUTER APPLICATION**

**Sastri/ B.A 3rd Year**
**Course/Paper.3**

# NATURAL LANGUAGE PROCESSING & PERL PROGRAMMING

राष्ट्रियसंस्कृतविश्वविद्यालयः, तिरुपतिः
(केन्द्रीयविश्वविद्यालयः)

**CENTER OF DISTANCE & ONLINE EDUCATION**

*(Formerly Directorate of Distance Education)*

**NATIONAL SANSKRIT UNIVERSITY :: TIRUPATI-517 507 (A.P)**

*(Erstwhile Rashtriya Sanskrit Vidyapeetha, Tirupati)*

# 1. (A) Introduction to Natural Language Processing

## Structure

---

**1.0. Objectives**

This chapter gives a brief introduction about Natural Language Processing (NLP), goals and its applications. It also explains the brief history of NLP and the knowledge sources required to build NLP applications. It also discusses the open problems in the present scenario.

**1.1. Introduction**

Natural Language (NL) is the language used by human beings as medium for the communication among themselves. Normally a communication process has the following steps

- the speaker encodes his ideas in NL text (synthesis)
- conveys this NL text to speaker through voice or written form
- listener understands this text (analysis to understand)
- listener responds the speaker (generation)

Encoding the our thoughts or abstract information into NL text is called **synthesis**.
Decoding or extracting the information encoded in the text is called **understanding**.
Analyzing the text to assign a suitable structure is called **analysis**.

These three processes analyzing, understanding, synthesizing the NL texts are called language possessing tasks.

When the whole or a part of the above process which helps to automate the language processing tasks, using computer, then we call this as Natural Language Processing (NLP). Technically, it can be defined as follows.

**Natural Language Processing (NLP) is a branch of computer Science which deals with the analysis, synthesis and understanding of natural language texts using computers.**

Therefore NLP is a discipline between computer Science and Linguistics (Linguistics is the study of NL dealing with the syntax and semantics of the language from Linguist perspective) and Logic.

The main task of NLP is to build **computational models** for the (a) analysis (b) generation and (c) understanding of NL texts.

Developing computer software systems that understand, analyze and generate NL text is a very difficult process due to the following reasons.

- ❖ Most of the words may have broad sense or ambiguous. The specific sense of the words depend on many factors like context, syntax etc.
- ❖ The meaning of the words depend on the speakers intention (vivaksha)
- ❖ Most of the time, the speaker does not code the entire information needed to convey. World knowledge or common sense of the listener guesses the information lost in the utterance.

Because many words have multiple meanings or senses which may depend on the structure and the context of its usage. Moreover, the speaker may not code the entire information which he intends to convey through his utterance. Certain information which is already known to the listener may not be coded. In such situations a great deal of world knowledge is required to understand the text. When human being is to analyze such text, different sources of knowledge like Language Knowledge (Grammar, Lexicon, real world knowledge), common sense knowledge (world knowledge, domain specific knowledge, context, culture knowledge) are used to understand the text fully.

So NLP programs require knowledge about language structure, its possible semantics (meanings) , an idea about the user of the utterance and a great deal of world knowledge. Hence when it comes to computerization of NLP activities, it becomes difficult.

**SAQ**

a. What is analysis?

b. What is synthesis?

## Need for studying NLP

1. To break the language barrier among the people of different countries and different languages
2. To build intelligible systems that help the man kind
3. To study how humans communicate using Natural Languages

### 1.2. Goals of NLP

The main objectives of NLP are

- To share the essence of the knowledge stored in the other languages using the known languages
- To develop cognitive and linguistic theories
- To do basic research on how humans communicate using Natural Languages.
- To build useful intelligent software systems

In order to achieve the above goals, the following sub tasks are required.

- Writing Computational Grammars (CG) for the natural languages
- Preparing corpora (Large amount NL texts which tell us how language is used)
- Preparing lexicons (Dictionaries)
- Search Engines for Indian Languages
- Building Machine Translation systems
- To build speech processing systems
- To build optical character recognition (OCR) systems

## 1.3. Applications of NLP.

The following are applications of NLP.

1. Natural Language Interfaces (NLI) to databases

   Nowadays most of the information is in the form of databases. In order to access it, one need not learn data base query languages. NLIs are used to interact with these databases in the natural languages to access the required information.

2. NLIs to computers

   If anybody wants to use computers for their day today activities, they must have the knowledge of Computer Operating system and a little bit programming knowledge. This is very difficult for an ordinary human being. So without knowing the Operating Systems & programming languages, one can operate comfortably computers using NLIs to communicate with the computers.

3. Question & answering systems

   Normal enquiry systems may be replaced with question answer systems which are provided with the domain knowledge enabling them to get answers for their queries.

4. Machine Translation systems(MTS)

   The richness of literature, scientific knowledge in one language can be shared by others in the languages known to them, using MT systems.

5. Text summarization

   Given any topic, vast information is available nowadays on internet. So there is a real need to summarize the information available there by reducing the time & effort of the user to make things understandable.

6. Knowledge Acquisition Systems:

   In order to develop intelligent computer systems which imitate human beings, a complex reasoning system must be evolved. It is possible only if well defined theories explaining how human acquire , store & access the knowledge are established.

7. Computer Aided Instruction/ Teaching (CAD/CAT)

Computers can also be used as a tool to teach the concepts in various applications.

8. Text to speech conversions and vice versa:

   Text to speech & speech to text systems are very much required as they are useful for (a) taking dictations, (b) hearing impaired & (c) Read for the blind.

**SAQ**

Specify the name of NLP application under which the following systems come?

a. Telephone automatic complaints receiving system ------------------

b. Tutoial CDs used for teaching ------------------------------------

c. Automatic translation systems on the net ---------------------------

d. Dictation taking systems on the computers ---------------------------

e. Automatic response systems at railway stations ---------------------------

## 1.4. Brief history of NLP

The field of NLP has emerged as merging of different disciplines like Artificial Intelligence (AI) of Computer Science, Linguistics, Logic, Philosophy etc. A large number of groups are working in each area. The major advances in the above fields have been brought together and this enhance research vigorously in NLP. The developments in the above areas listed here separately.

i) Linguistics: It has become the responsibility of the linguists to generate grammar formalism theories to analyse and generate NL sentences. So several theories of syntax were developed successively and derivation of the sentences from these theories were formalized.

Major ideas:

- Phrase Structure Grammar (PSG) by Gazdar
- Tree Adjoining Grammar (TAG) by Joshi
- Unification based Grammars by Kaplan & Brensman
- Indian Traditional Panini Grammar (PG) by Panini

The above theorems brought a major change in

- Parsing Algorithms
- Writing Comprehensive Grammars
- Dictionaries for machine use
- Statistical analysis of the language

b) Artificial Intelligence (AI): AI researchers attempted to look at the problem of language as that of communication. As a result all aspects of NL i.e., syntax, semantics, pragmatics are considered.

Knowledge representation and inference (reasoning) emerged as an important area, with a great significant contribution from AI workers in NLP.

c) Other Advances in the areas of Computer Science Other advances also influenced the NLP, some of which are listed below:

- Rapid increase in processing speeds
- Large memory
- Object Oriented Programming technologies
- Code generators
- Well developed Operating Systems
- Developments in building RDBMS
- Searching & sorting techniques
- Parallel Processing etc.

All the above advances are integrated in 1980's leading to the development of NLP (also known as computational Linguistics, CL)

**SAQ.** Write some Indian Old traditional grammars that you know or in your mother tongue.

## 1.5. Sources of knowledge that are useful in decoding the Natural Language :

There are several sources of knowledge that are used in decoding the information of a Natural language text, which are listed below.

1. Language Knowledge
   - Grammar
   - Lexicon
   - Pragmatics and discourse
2. Background Knowledge
   - General world knowledge or common sense knowledge
   - Domain knowledge
   - Context
   - Culture

A hearer can use all the above sources of knowledge to extract the information from a given source language text.

**SAQ**

Can you name the knowledge required to solve the following problems

a. To know whether the given noun is in nominative case -----------

b. To find the scope of slow in "slow protons and neutrons"---------------------------

c. To find out the meaning of "bank" ----------------------------

## 1.6. Open problems in NLP:

Although an impressive research had been made in NLP area, still there are many areas unsolved. Some of the issues are listed below:

- Knowledge representation
- Word Sense Disambiguation
- Natural Language Understanding
- Inference of a given natural language text
- The scope of adjectives & quantifiers
- Identifying cases (Karakas) and their roles to the noun participants in the sentence.

### 1.7. **Summary**

In this section, the following points are covered.

- The three processes viz., analyzing, understanding, synthesizing the NL texts are called basic language possessing tasks.
- NLP helps in overriding the language barrier.

- NLP is an interdisciplinary field of Computer Science, Linguistics, Logic etc.,
- The applications of NLP are of very much use to the mankind.
- Grammar Formalisms help to represent the language structure in terms of its meaning.

## 1.8. Model Questions

**15 Marks questions**

1. Give the introduction to Natural Language Processing.
2. What the applications of  Natural Language Processing.

**10 Mark questions**

1. What are the different knowledge sources used to decode the information uttered in a sentence?
2. Give Brief history of Natural Language Processing.

**5 marks questions**
1. Discuss the major open problems in NLP.
2. What are the basic tasks of NLP.
3. What are the basic tasks of NLP

| | |
|---|---|
| NLP stands for ----------------- | Natural Language Processing |
| NLI stands for ------------------ | Natural Language Interface |
| LIFER is a NLI system of -------- | Hendrix |
| INTELECT is a NLI system of --- | Harris |
| Unix is developed by ----------- | Wilensky at Berkeley |
| LUNAR is a question-answering system developed by ----- | Woods |
| Schank and Ableson did research on --- | Story understanding |
| Eurothra is MTS for ------ | European Languages |
| MU is MTS for ------ | Japanesh and English |
| Anusaraka is MTS for ------ | Indian Languages |
| ACL stands for ----- | Association of Computational Linguistics |

| | | |
|---|---|---|
| **1 mark questions** | Semantic theories were proposed by ---- | Fillmore, 1968 |
| | LFG stands for ------ | Lexical Function Grammar |
| | LFG was proposed by ----- | Kaplan and Bresman |
| | GPSG stands for --- | Genaralized Phrase Structure Grammar |
| | GPSG was proposed by ---- | GAdzar |
| | TAG stands for --- | Tree Adjoining Grammar |
| | TAG was proposed by --- | Joshi |
| | FOL stands for ---- | First Order Logic |
| | Lexicons are concerned with ----- | Dictionaries |

# 1. (A) Language Structure and Language Analyser

## Structure

### 2.0. Objectives

In this section, we will learn how to analyse the given NL text. We also learn how words combine together to form a structure. we will also learn the general structure of a language analyser and its related tasks.

### 2.1. Introduction

A language analyzer is a system, which finds out the internal structure of a given sentence and stores this structure in terms of as attributes. Hence the analysis of natural language texts deals with the extraction of information encoded in the given text. The process of analyzing a given word and assigning a suitable structure is called parsing. The parsing is to be done at three levels. They are

| **Level** | **Information encoded** |
|---|---|
| Word | Word root, category and other grammatical feature |
| phrase | Identification of the combination of two or more number of words which contribute the meaning that con not be derived from the combination of the given words |
| Sentence | The overall structure and the meaning of the given sentence |

In this unit we will study how to analyse the given text of any language at word, phrase and sentence levels.

## 2.2. Language Structures

The basic unit of any text is a word. Each word carries some information. When such words are grouped together, they form a word group. There is a systematic way to identify the information contained in the word group. When two or more number of words combined to form a word group, a structure is formed. The information contained in the word group depends on

- the way in which words are grouped together
- the meaning of the individual words
- the relation among words of the structure

To keep the structure minimal, **modifier-modified relation** is introduced among the words in the word group. The word which modifies all other words in the group is called modifier; and the word modified is called the head word.

| Ex. | Head | Modifier |
|---|---|---|
| | Fake | gun |

In the above example, the properties of the entire word group are derived from the head word, with attributes of the modifiers.

The modifier – modified structure can be broadly classified as nominal and verbal structures.

### 2.2.1. Nominal Structures

In these structures, noun is the head word resulting the entire structure to be nominal. The different kinds of nominal structure are given below.

1) **Nominal structure with adjective–noun modification**

Here noun is the head and the adjective is the modifier.

Ex:

| Adjective (modifier) | Noun(Head) |
|---|---|
| White | flower |
| Fat | boy |

2) **Nominal structure with participle verb as modifier of noun**

In this case, the noun is head and non-finite or participle form of the verb will be the modifier.

Ex. Non-finite/ Participle form of verb noun

| | |
|---|---|
| Eaten | fruit |
| Running | deer |

3) **Nominal Structure with verbal noun**
A verbal noun behaves like a noun but many of its properties are maintained by the verb. The modifier-modified relation can be identified for this, as usual for other verbs.
Ex.
Eating fruits ( is good for health).
Listening music ( consoles your mind).

**SAQ**
Name and list some nominal modifier-modified structures in your mother tongue

**2.2.2. Verbal Structures:**

Verb is the head in these structures, resulting the entire structure to be verbal. However, verb denotes the action performed by noun entities. So verbs are modified by noun participants.

1) **Verbal Structure with verb-noun modification**
Different linguists give different accounts of modification i.e., case grammars, thematic roles, agents etc. But Panini theory deals in a separate theory called Karaka theory. Karaka theories identify relations between nouns and verb in terms of Karaka.

Ex. Rama killed Ravana.

$$\begin{tikzcd}
 & \text{Kill} \arrow[dl, "\text{Karta}"'] \arrow[dr, "\text{Karma}"] & \\
\text{Rama} & & \text{Ravana}
\end{tikzcd}$$

2) **Verbal structure with verb-verb modification**
Here one verb is the head and the other is the modifier verb. The modifier verb indicate that its action occur before or during or after the main action.
Ex.1. Having eaten the rice, he went to Madras.

2. He went to Madras while eating the rice.

In all the above cases, went is the main action and is the head of sentences. 'Eat' is the sub action, modifying the went(main) action.

3) **Verbal structure with verb as the argument of the head verb**
One verb become the argument of the another verb in this case.

For example ,

Rama told Bhima to eat the cake.

In the above case, 'told' is the main head verb for which 'eat' becomes the argument.

**SAQ**

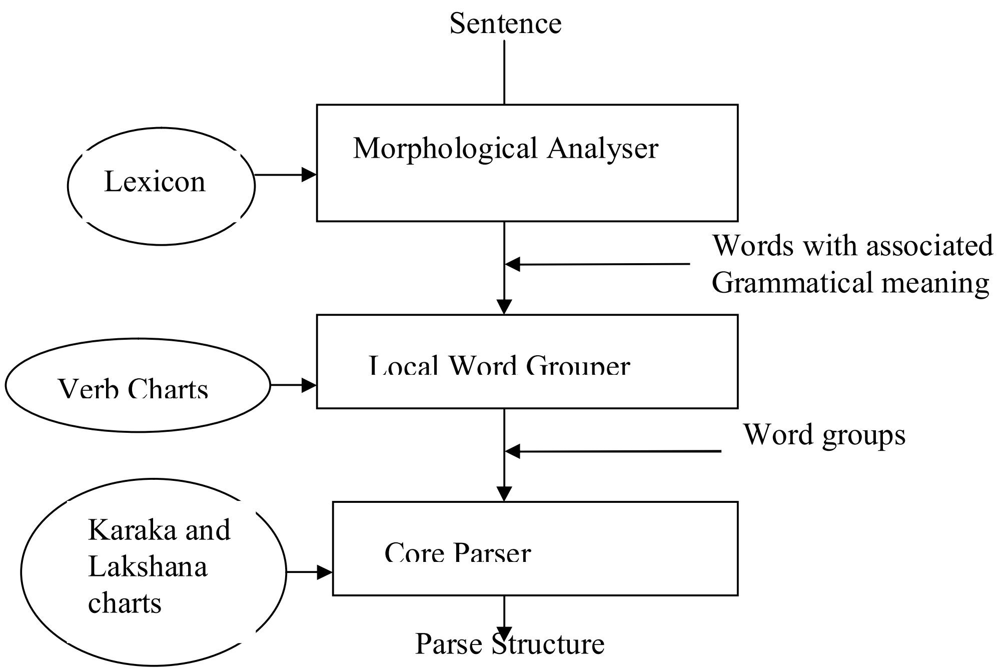
Name and list some verbal modifier-modified structures in your mother tongue

## 2.3. Overview of the language analyzer /Parser

The main function of language analyser is to analyze the input text (normally a sentence), and assign a suitable structure. This process is known as parsing. Therefore, the parser takes text as input ( a group of words) uses grammar of the language, produces a structure. This structure is known as parse structure and contains the complete information of text in terms of attributes of each word, group of words which give some collective meaning and the relation among words of the sentence. So parser or language analyzer consists of two components:

1. Procedural component and 2. declarative component

By changing the grammar the procedural component would parse different language. How ever the above two components depend on the grammar formalism which help us to derive the parse structure. The grammar suitable for Indian Languages is the Panini Grammar (PG).



**Morphological analyzer**: It takes a sentence as its input. Each word of the sentence is associated with a structure indicating the root of the word, its lexical category, gender, number, person, tense ( for verbs only) etc. To achieve this a dictionary or lexicon needs to be looked up to retrieve the above information. Some times analysis of the word is also required.

For example, assume that the input sentence:

| | | |
|---|---|---|
| rAmaH | rAvanaM | mArayawi |
| Rama | Ravana | Kills. |

Here the Sanskrit sentence is transliterated using roman coding scheme. See the Appendix-1 to learn to transliterate the Indian Script into Roman Coding Scheme.

The output of the morphological analyzer for the above sentence is given below:

rAmaH :

| | |
|---|---|
| Root | : rAma |
| Lexical Category | : noun |
| Gender | : male |
| Number | : singular |
| Person | : third |
| Case | : prathama |

rAvanaM:

| | |
|---|---|
| Root | : rAvana |
| Lexical Category | : noun |
| Gender | : male |
| Number | : singular |
| Person | : third |
| Case | : Dwitiya |

mArayawi:

| | |
|---|---|
| Root | : mq |
| Lexical Category | : verb |
| Number | : singular |
| Person | : third |
| Tense | : present |

**SAQ**
What is the morphological output for the following words
a) rAmena

b) hasAmi

**Local word grouper**: The function of this block is to form word groups on the basis of the local information. For example,

English: Look after, stand up, in front of, in stead of, etc.

Telugu:: pelli kUwuru, polamu putra etc. (noun groups)
pelli ceSAdu, prayAnamu ceSAdu (verb groups)

Hindi : roTi kapaDA makan (noun groups)
SaxI karanA, kara sakawA hE (verb groups)

In above cases, words should be treated as a single unit and hence they are to be grouped together to get the correct analysis.

**SAQ**
Can you list some noun groups in your mother tongue?

Can you list some verb groups in your mother tongue?

**Core Parser**: The function of core parser is to accept the outputs of previous blocks and to assign a suitable structure. The main function is to get Karaka relations between verbs and nouns.

For example, the structure of the parser for the sentence

| | | |
|---|---|---|
| rAmaH | rAvanaM | mArayawi |
| Rama | Ravana | Kills. |

Mq (kill)

Karta : rAma
Karma : rAvana

**SAQ**

What would be the out put of the following sentence?
hariH vanaM gaccawi.

## 2.4. Characteristics of NLP Systems

The important objectives of any NLP application are:

1. analyzing the given input NL text and
2. generating NL text from the given inputs

So NLP consists of grammar formalisms in addition to analysis and synthesis programs which actually use grammar of the language to be analyzed and synthesized. The NLP programs should satisfy the following characteristics or requirements.

1. Computational aspect: Grammar formalisms should consider the regularity of the word forms in such a way that they can be analysed and generated computationally.
2. system aspect: The system to be developed should consider all the aspects of the problem, giving importance to even minute detail, and it should be built according to software engineering principles.
3. Large system aspect: Most of the NLP systems are complex and large. So the following properties should be satisfied.

(i) Modularity: The problem is to derived into sub problems and a separate module is to be developed for each sub problem. All these modules are to be integrated properly is such a way that they can be upgraded individually, if the need arises.

(ii) Extensibility: The system can be extended which may be due to the following reasons.
   a. Technology
   b. New Software packages
   c. New theories in grammar formalisms
   d. increase in scope of problem'

(iii) The system should be capable of dealing with failures

(iv) Sufficient feedback must be provided to fix or debug the problems

(v) When the system has to perform a task beyond its scope, it must handle them properly without being lost.

(vi) The system should be tolerant of errors.

### 2.5.Summary

In this sub-section, we have studied about
- how words combine together to form modifier-modified word structures
- general overview of the language analyzer
- general characteristics of NLP applications

## 2.6. Model Questions

**15 Marks questions**

1) Define modifier-modified structure. Explain different modifier-modified structures with suitable examples.
2) Give an overview of Language Analyser.

**10 Mark questions**

1) Discuss the verbal modifier-modified structures.
2) Discuss the nominal modifier modified structures..
3) Explain the modifier-modified structure for the following sentences.
   a) Ram asked Mohan to read the book.
   b) The child wants to go home running.
   c) Ram is feeling cold.
4) Discuss the requirements of Computational Grammar

   Or

   Discuss the characteristics of NLP systems..

**5 marks questions**

1) Write noun-verb modifier-modified structure for the following sentence.

   Ram kills Ravana with an arrow at Lanka.
2) Explain adjective-noun modifier-modified structure with suitable example.
3) Explain adjective-noun modifier-modified structure with suitable example.
4) Explain verbal structure with noun-verb modifier-modified structure with suitable example.
5) Explain nominal structure with participle as modifier of noun with a suitable example.

**1 mark questions**

| | | |
|---|---|---|
| 1. | Lexicons are concerned with ----- | Dictionaries |
| 2. | The two items present in the modifier-modified structure are ---- and --- | Head and Modifier |
| 3. | In 'naughty boys' , ----- is the modifier and --- is modified | Naughty and boys |
| 4. | 'fat boy' is an example of ----- modifier-modified structure | Adjective-noun |
| 5. | 'Running boy' is an example of ----- modifier-modified structure | Participle verb as modifier |
| 6. | Give an example for verbal noun | Doing exercises - English<br>wAgadaM – Telugu<br>KAxanaM – Sanskrit |

| | | |
|---|---|---|
| 7. | ----- analyses and assigns a suitable structure to each word of the sentence | Morphological Analyser |
| 8. | Morphological analysis for 'rAmeNa' is ------ | Root – Rama<br>Category –noun<br>Gender – Masculine<br>Number – singular<br>Person – Third<br>Case – 3 |
| 9. | ------ forms word groups based on local information | Local Word Grouper |
| 10. | Paninian Core Parser makes use of ----- and ----- principles. | Akanksha and Yogyata. |
| 11. | The eligibility of nouns to satisfy the demands of verbs is called ------ | Yogyata |
| 12. | The demand of verbs for nouns to be their karakas is called ----- | Akanksha |
| 13. | -------- specifies the relation between two constituents when they are close to each other. | Sannidhi |
| 14. | An NLP system consists of ----- and ------ | Grammar and Procedural component |
| 15. | The grammar formalisms that have been designed with the processing of mind are called --- | Computational GrammarFormalisms |
| 16. | ------- says that 20 percent of grammar covers the 80 percent of the language. | 80-20 rule |
| 17. | ------ implies that a large program can be divided in to several parts in which interaction is minimal | Modularity |
| 18. | ------ means that the the system can be extended task by task. | Extensibility |

# 2. (A) WORD AND THEIR ANALYSER

## Structure

---

### 3.0 Objectives

In the last section we have studied about the analysis of a sentence. In this section, we will learn how to analyse words of the given text to assign a suitable structure to each word of a given language. We also learn that the process of analysing a word is called Morphological analysis.

### 3.1. Introduction

Morphological analysis (MA) deals with the segregation of word into subunits called morphemes. Morphemes are meaning bearing small units. By identifying the morphemes of a given word, we will be able to derive the form (syntax) and meaning (semantics) of the word analysed.
In this section we will study how word and its affixes give different structures to the given word. Also the general structure of the word analyzer (MA) is introduced.

### 3.2. Words and their types

Word is a sequence of characters delimited by space. Word in any language may have two forms.

(1) Simple word is of the form root plus affix (prefix or suffix). Affixes can not appear independently and are known as morphemes.

For example, consider the word "rAmena"
Here root is "rAma"
Affix is "ena"
Are you able to understand that as the affix changes the meaning of the word also change?

**SAQ**
Give one simple word in your mother tongue. List its root and affixes.

(2) Compound word consists of two or more simple words.
For example, consider the word "padakamalam".
Here the word is a combination of "pada", "kamala" and "aM'

**SAQ**
Give one compound word in your mother tongue. List its roots and affixes.

## 3.3. Morphological analysis

Word analysers should analyse the given word into its root, affixes and feature values of the grammar like number, gender, person etc. The process of analyzing the given word to extract the information encoded in the text is called as morphological analysis.

**The need for word analyser**:

If analysis is not done, all the possible word forms should be stored exhaustively. It is not possible due to the following reasons.

a. It is a memory wastage.
b. Linguistic generalization is not shown, leading to redundancy.
c. The language has rich and productive morphology.

However, there should be a balance between storage of word forms and analysis

The linguist should provide word form table which gives all the forms of the given word. When we consider all the words of the language, roots of certain words produce its word forms in the same fashion. For all such forms definite paradigm tables PT( tables which show all the forms of words together with feature values which vary with the word form) and store them appropriately. Dictionary of roots (DR) storing all the root words exclusively with feature values (which are fixed with the root word) are also to be stored. The roots which do not behave uniformly with other words are stored in dictionary of indeclinable (DI), with all its forms along with the feature values associated with it.

For example, consider the following words which are listed in the table below.

Dictionary of roots (DR):

| Root | Category | gender | Person | Paradigm type |
|---|---|---|---|---|
| LaDakA | N | male | 3 | laDakA |
| KapaDA | N | male | 3 | laDakA |
| BASA | N | female | 3 | BASA |
| LaDakI | N | female | 3 | laDakI |
| ROTI | N | female | 3 | laDakI |

Word form table (WFT):

Root : laDakA

| Number | Case | |
|---|---|---|
| | Direct | Oblique |
| Singular | laDakA | laDake |
| Plural | laDake | laDakoM |

Paradigm table (PT):

Root : laDakA

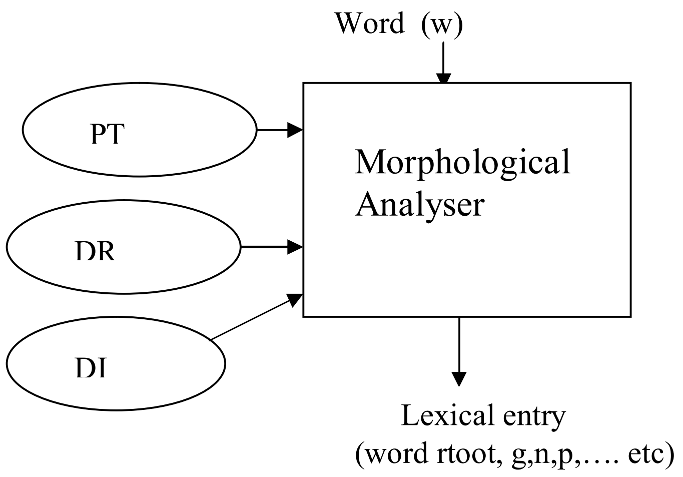
| Number | Case | |
|---|---|---|
| | Direct | Oblique |
| Singular | (0,Φ) | (1,e) |
| Plural | (1,e) | (1,oM) |

Since kapadA behaves as laDakA in forming word inflections, we need not store the paradigm table for kapadA. Hence we listed in the DR table that the paradigm type of kapaDA is laDakA.
LaDakI, roTI behave in the same way to form word inflections. Hence it was indicated in the DR table.
Since BASA has no matching word in the DR table, we can have another dictionary called Dictionary of Indeclinable (DI) , in which all inflections of BASA along with their features can be specified.

| Word | Word feature |
|---|---|
| BASA | Singular Direct |
| BASA | Singular Oblique |
| BASAeM | Plural Direct |
| BASAoM | Plural Oblique |

The paradigm table can be extracted from WFT by identifying the number of characters to be deleted from the root and the characters to be added to obtain WFT.

**Block diagram of Morphological Analyzer (MA)**



Algorithm:

```
1. Let L be the output file record with a structure
2. If word (w) is an entry of DI, add this entry to L
3. for i = 0 to length(word) do
     Let s = suffix of word of length i
     for each paradigm table P
```

```
            for each entry (j,a) in P do
                if (s = a)
                {
                    r   = root of the paradigm table P
                    pr =  (word – s) + suffix of r of length i
                    if (r is in DR along with paradigm_type P)
                    {
                        add   required features for DR plus P
                    }
                }
            endfor [ each entry]
        endfor [paradigm table p]
      endfor [i]
4.  If L is empty, print  " unknown word"
    Else   return L
```

Explanation of algorithm:

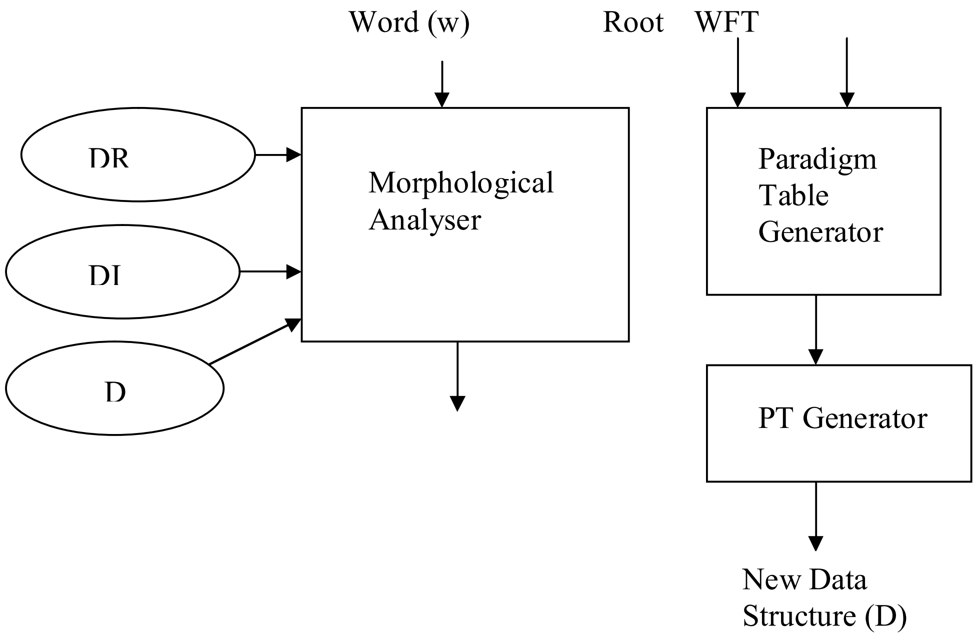
Suppose the given word (w) is laDake and we are asked to analyze it. The analysis should tell the root of the given word along with its feature values. Now we have two three PTs.

The first step is to check whether it occurs as an indeclinable word in DI. The next step is to check all the entries in all PTs, whether the last suffix of laDake matches with all entries of PTs. Then by deleting and adding as many as characters as shown in PT, we get all possible roots. The roots together with paradigm type in DR are checked with the possible root in consideration to find the correct roots of the word.

**SAQ**
Try to write word form table and paradigm table for one word in your mother tongue.

## 3.4. Speeding up morphological analysis by compiling paradigm tables.

The method described above is quite expensive. Each entry in every PT is compared for every possible suffix. This search can be speeded up by compiling PTs before hand and generating a data structure D, which is used by morphological Analyser in stead of PTs directly.



The key idea is that the suffixes are ordered and so the original suffixes are ordered and so the original paradigm is not needed. Given a suffix, it can be located very easily without scanning all the entries in PTs.

Ex. sorted Reverse suffix table (RST)

| Reverse Suffix | Character string To be appended to the prefix | Grammatical features (gender, number, PT type) |
|---|---|---|
| Φ | Φ | s,d,laDakA |
| Φ | Φ | s,d,BASA |
| Φ | Φ | s,o,BASA |
| E | A | s,o,laDakA |
| E | A | p,d,laDakA |
| Me | Φ | p,d,BASA |
| Mo | A | p,o,laDakA |
| Mo | Φ | BASA |

Algorithm:

```
1. Let L be the output file record with appropriate structure
2. If word (w) is an entry of DI, add this entry to L
3. for i = 0 to length(word)   do
       let s = suffix of w of length i
       if reverse s is in RST
           for each entry b with an entry reverse suffix s in RST do
                   root = w- suffix s + string to be added from the root
                   if root is in DR with PT as entry b
                       add b to L
                   endif
            endfor
         endif
     endif
4. if L is empty then return "unknown word"
     else
       return L
```

**3.5. Summary**

In this section, we have studied about

- What is meant by Morphological Analysis
- The need for Morphological Analysis
- How to prepare data (Dictionaries) like WFT, PT, DR, DI etc for Morphological Analysis.
- How to write programs of less complexity for analysis of the words of a given language
- How to speed up Morphological Analysis

**3.6. Model questions**

**15 Marks questions**

1. Give an overview of Language Analyser.
2. Explain the algorithm for Morphological Analysis using Paradigm Tables.
3. Explain how the morphological analysis can be speeded up.

**10 Mark questions**

1. Give Word Form Table for "Rama" in Sanskrit and explain how paradigm table can be constructed for the same word
2. Define the following terms
   a) Morphological Analysis
   b) Word Form Table
   c) Paradigm Table

**5 marks questions**

1. Give Word form Table for "Rama" in Sanskrit.
2. Why Morphological analysis is done using Paradigm Tables.
3. Explain the structure of Dictionary of Roots.
4. Explain the structure of Dictionary of Indeclinable.

**1 mark questions**

| | |
|---|---|
| 1. Word can be of ----- and ------ types | Simple and Compound |
| 2. Constituents of simple word is called a ----- | Morpheme |
| 3. ----- covers a the patterns of a given root word. | Word Form Table |
| 4. WFT stands for -- | Word Form Table |
| 5. PT stands for ---- | Paradigm table |
| 6. Paradigm table is created from ---- | Word From Table |
| 7. RST stands for ----- | Reverse Suffix Table |

# 2. (B) LOCAL WORD GROUPING

## Structure

### 4.0. Objectives

In this section, we will learn what does it mean by local word grouping. The need for local word grouping is thoroughly discussed. Also noun groups and verb groups are introduced.

### 4.1. Introduction

Some times, the meaning of individual words can not be taken into consideration in the process of analysis, to get the overall meaning of the sentence. This is due to the reason that the phrase (a group of words) must contribute the other meaning other than the combined meaning of the constituent words. For example, the phrase "in front of" in English can not be individually analysed. In this section, we will learn how to identify and process such word groupings.

**SAQ**
Can you list such words in your mother tongue?

## 4.2. Local Word Grouping.

The meaning of a word group or a collection of words generally depends on

- the meaning of individual words and
- the relationship (syntactic and semantic) among words of the group.

However this collective meaning sometimes can not be derived by the individual words of the word group. The reasons for this are given below.

1. Individual words of the group can not be used independently. (Ex. Vibhaktis in almost all languages, as they precede or succeed with nouns)
2. A group of words act like an idiom or phrase, giving separate meaning which is not related to the individual words of the group. (Ex. Pelli kUwuru in Telugu)
3. For some languages like Hindi, verb and TAM (Tense, Aspect, Modality ) are written as separate words.

In such situations words need to be grouped together. Generally these word groups may come under two categories viz., noun and verb groups**.**

### 4.2.1. Noun groups

Noun groups may be formed as shown below.
**(a) nouns and vibhaktis**
Normally noun inflections in any language are a combination of noun root stem and vibhakti (case), indicating the role of the noun (like karta, karma etc). Hence they can not be analysed separately. In some languages like Sanskrit, they appear as a single word. But in languages like Telugu, they may appear as separate words. In such situations, they are to be grouped.
Ex. rAm kA, rAm ne, rAm se in Hindi, rAmuni valana in Telugu

**SAQ**
Can you list such words in your mother tongue?

**(b) nouns and nouns**
All most all languages have compound nouns. They also are to be grouped. For example,

Fruit flies, House flies in English
pelli kUwuru, nagA natrA in Telugu

**SAQ**
Can you list such words in your mother tongue?

**4.2.2. Verb Groups**

Generally words containing verb may indicate the following

a. Root of the verb indicating action
b. Tense indicating when action occurs
c. Gender, number, person of the Karta or Karma

In certain languages they are written apart like in Hindi (Ex. Ka raha Hai) and in certain languages they are written as a group like single unit (namAmi in Sanskrit). In the first case all the words are to be grouped together.
Kriya rupa chart in Hindi specify the groups to be formed out of the sequence of verbs which denote a single action. For example,
jA raHA HE
KA raHA HE

In certain languages like Telugu, the combination of some verbs with some nouns give different meaning altogether.
Ex. kallalO nippulu pOsukonnAdu. (felt gelous).

Therefore, two or more words are to be grouped together to derive the original meaning of the word group and this is known as local word grouping.

**SAQ**
Can you list such words in your mother tongue?

## 4.3. Some open problems in Local word grouping

We list below some problems relating to conflicts between parsargs and other lexical categories, for which the solutions in terms of suitable rules remain to be worked out.

1. some time vibhaktis get overloaded with many karakas. For example,

   a. Hindi

   'se' can be used as a comparision word as given below

   **Phoola se komala raama ne dhanusha toda diya.**

   If 'se' is taken as a parsarg here, the sentence would indicate flower as an instrument:

   Perhaps when 'se' is used as a comparision word, there are only a limited number of word that can follow it. Linguists can make a catalogue of such words.

   b: English

   **He came by 4'O clock.**

   In English, normally the word 'by' is used for instrumentation (karana karaka). But here, it is used to indicate locative (adhikarana)

2. Sometimes the same word belongs to different parts of speech.

   a. Hindi

   **raama ne mohana se baata kii.**

   'kii' can occur as a verb as well preposition (vibhakti) .

   It might appear that one way to distinguish between the two usages is that the verb occurs at the end of the sentence unlike the parsarg. When 'kii' occurs at the end, it can only be a verb. But would the following sentence be considered an incorrect sentence:

   b: English

   **The books are red.**
   **He books the tickets.**

   'books' can act as both noun and verb. Depending on the sentence structure, we can decide whether books is noun or verb. In first sentence, the words "The books" are to grouped. In second sentence, 'books' need not be grouped with any other word in the sentence. However, observe that the words , 'the tickets' are to be grouped together.

## 4.4. Summary

In this sub-section, we have learn
- the concept of local word grouping
- different types of local word grouping
- the problems associated with local word grouping

## 4.5. Model Questions

**15 Marks questions**
1. Write a short note on local word grouping.

**10 Mark questions**
1. Discuss Open problems in Local word grouping

**5 marks questions**
1. What do you mean by a verb group
2. What do you mean by a noun group.

**1 mark questions**

| | | |
|---|---|---|
| 1. | Indian Languages have --- word order | Free |
| 2. | ---- specify the groups to be formed out of the sequence of verbs which denote a single action | Kriya Rupa Charts |
| 3. | Noun groups are formed out of nouns and ---- | Vibhakti |
| 4. | Utsarga-apavada (default-exception) apprach was proposed by ---- | Patanjali |
| 5. | The approach wherein the grammar rules are written in layers, each layers containing rules and forming an exception to the higher level is called ---- approach. | Utsarga-apavada |
| 6. | Nouns have less inflections in ---- and --- languages. | English and Hindi |

# 3. PANINI GRAMMAR

## Structure

### 5.0. Objectives

In this section, we will see how to derive a meaning of the sentence based on Karaka theory, as dealt in Panini Grammar, the Indian Tradition Grammar. We see how overall computational model can be developed to parse the entire sentence for a given language.

### 5.1. Introduction

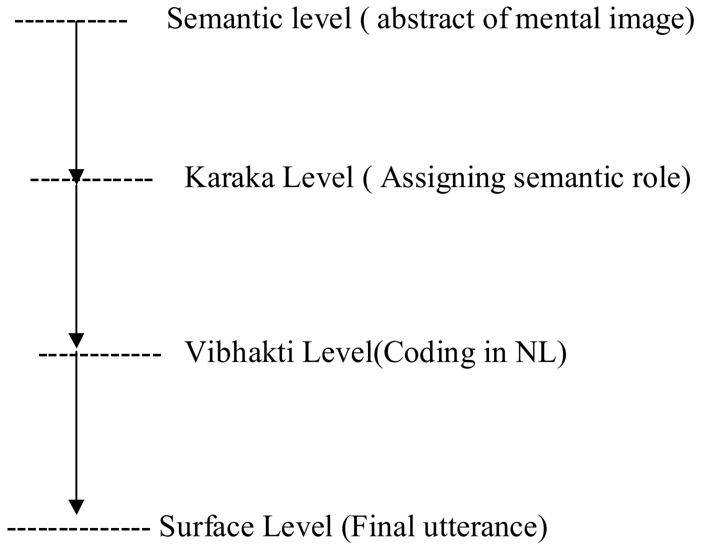
Grammar explains how to understand the meaning of a given sentence. Also it explains how to produce an intended sentence. In the context of language processing, we are interested how to analyse and synthesize natural language texts. There are many grammar formalisms which explain the syntax (sentence structure) and semantics (meaning) of natural languages. But no other grammar formalism is so algorithmic in nature which can explain the whole language phenomenon as Panini grammar does. Hence we are studying Panini Grammar in this section.

### 5.2. Panini Grammar.

The main objective of NLP is to extract the information and meaning in the Natural Language text. Many approaches and theories are developed to extract the semantic content of NL text. One among them which is more suitable to Indian Language is Panini Grammar. The main feature of Indian Language is word free order.

Panini Grammar can be described by the following points.

1. Each and every action has many nominal participants.
2. Each verb refers to an action which has
   - (a) phala (result) and
   - (b) vyapara (activity).
3. Each action can have sub activities. Each activity has its own participants.
4. The speaker has vivaksha (intention/opinion) about the listener and this is conveyed in his uttering.
5. Action springs from karta and is experienced by the karma nominal, which is known as sakarmaka (transitive).
6. If the action is experienced by the karta itself, it is known as intransitive (akarmaka).
7. Karaka relations are the relations between nominals and verbals.
8. Karaka relations are syntactico-semantic relations. This notion is explained below.


Semantic level ( abstract of mental image)
Karaka Level ( Assigning semantic role)
Vibhakti Level(Coding in NL)
Surface Level (Final utterance)

There are six different types of karaka relations which explain the relationship of action and its participants. However it is not clearly possible to represent all semantic relations between nouns and verbs of the sentence. But these karakas sufficiently explain the relations among verbals and nominals of the sentence.

**5.3. Karakas**

The six karakas are explained below.

1. Karta Karaka: The most independent participant in the action.

Ex. Ram reads.

2. Karma karaka: Ashraya of the result of the action.
   Ex. Rama kills Ravana
3. Karana karaka: Instrument with which action is performed.
   Ex. Ram writes with pen.
4. Sampradana Karaka: is the beneficiary participant in the action.
   Ex. Ram gives fruit to John
5. Apadana Karaka: The nominal which remains stationery when separation takes place.
   Ex. Flowers are falling from the tree
6. Adhikarana Karaka: specifies time and location.
   Ex. He went there at 4'O Clock.

**SAQ**
Can you write one sentence in your mother tongue and identify the karakas for all nouns.

The problems with **Vibhakti – karaka** mapping are:

1. Different Vibhaktis may represent the same karaka information.
2. Same Vibhaktis can be used to represent different karaka information.

The important insight regarding the karaka-vibhakti mappring is that it depends on the verb and TAM (Tense, Aspect, Modality of the verb-concatenation of main verb, raw terms of auxiliary verbs). This mapping can be represented by two structures.

1. Default Karaka Chart: This gives Karaka chart for a specific verb or class of verbs for a specific TAM Table.
2. Karaka Chart transformation: for other TAM tables of same verb, the transformation rules can be written in the transformation chart.

For example, default karaka chart for three karakas for ***taa-hei*** is given below.

| Karaka | Vibhakti | Presence |
|---|---|---|
| Karta | φ | Mandatory |
| Karma | ko or φ | Mandatory |
| Karana | se or dvAra | optional |

Transformation rules for other forms of the verb:

| TAM label | Transformed vibhakti for karta |
|---|---|
| yaa | ne |
| naa_paDaa | Ko |
| Yaa_gayaa | se or dvArA |

**SAQ**

Can you write a default hart and transformation rules for a verb in your mother tongue?

## 5.4. Karaka Sharing

When two or more number of verbs are used in a sentence, Panini theory supports elegantly the karaka roles through karaka sharing.

The main question is which verb controls the vibhakti the vibhakti and karaka sharing. The answer can be explained in two points.

1. Karaka to vibhakti mapping
   a. The intermediate verbs also have TAM Tables like other verbs.
   b. Therefore TAM Tables have transformation rules that modify and operate karaka charts.

2. Karaka sharing

   In order to obtain shared karaka rules, the following rules for Hindi may be used:

Now we give rules for obtaining the shared karakas.

Consider the following sentences in Hindi.

a. rAma Pal KAkara mohana ko bulAwA hai
b. rAma ne Pal kATakara khayaa
c. Phala kATane ke liye usane caakuu liyaa.

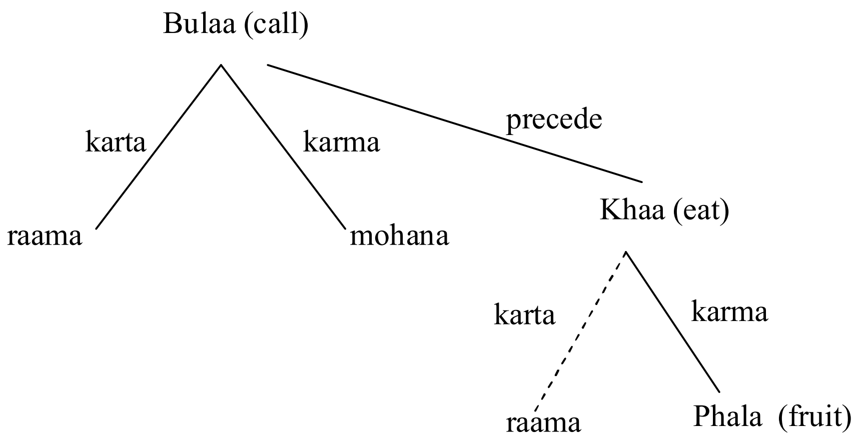
Karta of the intermediate verb khaa in a and b can be obtained by the sharing rule S1' which given below.

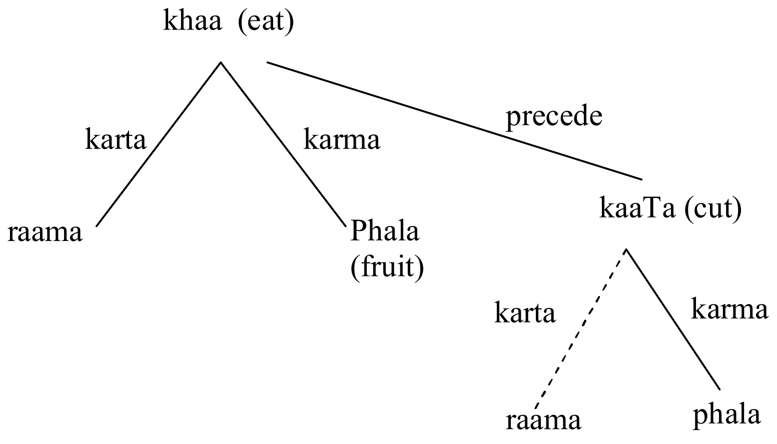
Rules S1' : Karta of verb with TAM label 'kara' is the same as the karta of the main verb.

The sharing rule(s) are applied after the tentative karaka assignment is over using karaka to vibhakti mapping.

For karma of intermediate verb with TAM label kara, we have the sharing rule S2'.

Rule S2': If an intermediate verb with the TAM label 'kara' takes a karma (as specified in its default karaka chart) while none has been obtained using karaka vibhakti mapping, then it shares its karma with the karta or the karma of the main verb.





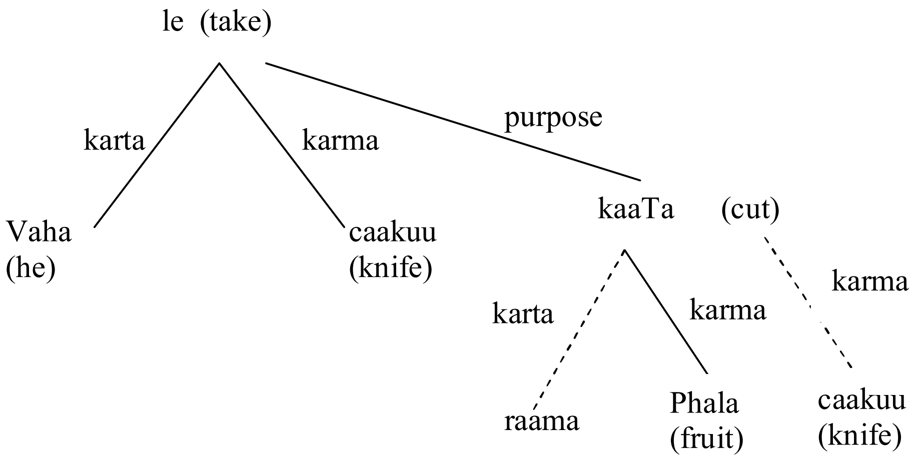


Figure -a

With the above rules, the shared karaas for a,b and c are shown in figure-a by dotted lines. In the following example,

| | |
|---|---|
| d. | khariidakara raama ne phala khaayaa. |
| | Having-bought Ram –ne fruit ate |
| | (Ram ate fruit having bought it.) |

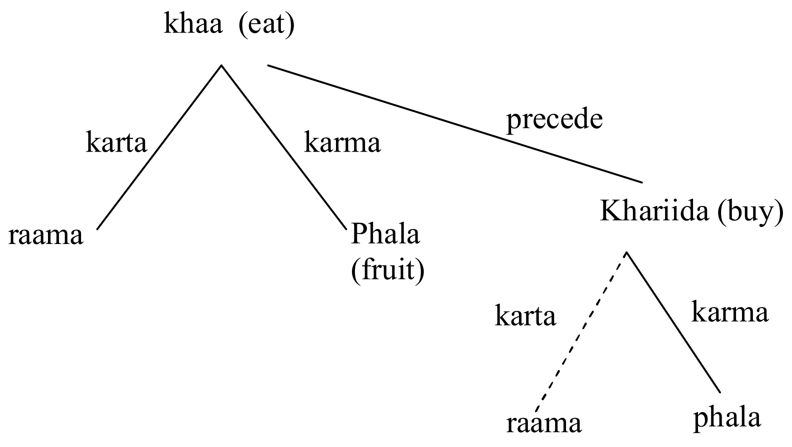


Figure-b : parse structure for sentence d

karta and karma of khariida (buy) are the same as the karta and karma, respectively, of khaa (eat). Figure-b shows the relationships pictorially.

Similarly, the sharing rule for taa_huaa can be stated as S3'.

Rule S3': An intermediate verb with TAM lable 'taa_huaa'shares its karta with the karta of the main verb.

As example, consider:

e. shikaarii ne bhaagate hue shera ko dekhaa.
Hunter –ne while-running lion –ko saw
(The hunder saw a lion while running.)

The above is ambiguous between whether the hunter was running or the lion was running. Now, karaka vibhakti mapping yields the following relation:

Dekha (see)
    Karta:shikaarii (hunter)
    Karma:shera(lion).
Bhaaga (run)
    No karaka relation expressed explicitly.

The above are shown pictorially in Figure -c(ignore dotted lines for now).

By Rule S3', we can identify the shared karakas of run. Here, the karta of run would be the hunter. This is shown by dotted lines in Figure -c.

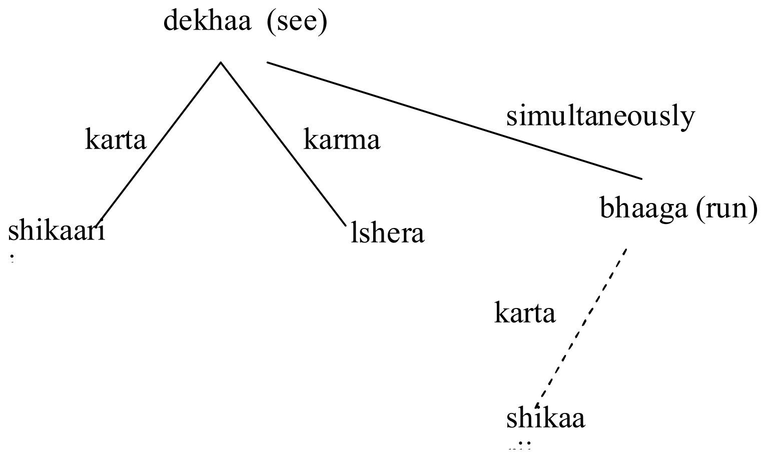


Figure-c : parse structure for sentence e

There is another parse for the sentence corresponding to verb-noun modification with verbal 'bhaagate hue'(running) modifiying noun 'shera'(lion). Karaka relations of dekha (see) as obtained by karaka vibhakti mapping are the same. But since bhaaga modifies shera, the karta of bhaaga is hsera by rule S4 given below.

Rule S4: If a verb with TAM label taa_huaa modifies a noun, then that noun is its karta.

The part of parse structure shown by solid lines is obtained by karaka vbhaki mapping. The part shown in dotted lines (the karta of bhaaga is obtained by Rule S4.

When there are more than one intermediate verb groups in a sentence, there is a need to generalize the rules given earlier. To obtain the shared karaka of an intermediate verb, instead of getting it from the main verb we must obtain it from the verb(action) modified by the intermediate verb. The modified verb could be the main verb or another intermediate verb.

## 5.4. Summary

In this chapter, we have studied about
- how Natural Language texts can be understood based on Panini theory.

- Karakas
- vibhakti –karaka mapping
- how to prepare verb charts for a given standard verbal inflection
- how to verb transformation charts for non-standard verbal inflections of a given verb
- how karakas can be shared when there is more than one verb in the Hindi sentence

## 5.6. Model Questions

### 15 Marks questions

1. Write a short note on Panini Grammar.
2. What are the problems with vibhakti to karaka mapping? How they could be solved

### 10 Mark questions

1. Write a short note on Karaka sharing.
2. Explain the six Karakas with suitable example sentences

### 5 marks questions

Identify the karaka roles for the following sentences.

1. Rama killed Ravana with an Arrow.
2. He gave a book to Sita.
3. While walking, he did shopping.
4. He came by 4 O' Clock.
5. He ate bread pieces in the salad.

### 1 mark questions

| | | |
|---|---|---|
| 1. | The relations between noun and verbs are indicated by ---- in Indian Languages | Karakas |
| 2. | Karakas are denoted through ---- in Indian Languages | Vibhaktis |
| 3. | In Panininan framework, every verbal root denotes action consisting of ---- and ----- | An activity (vyapara) and a result (Phala) |
| 4. | The nominal participants of an action carried out in a sentence are called --- | Karakas |
| 5. | ------- refers to the speaker's viewpoint or attitude towards the activity. | Vivaksha |
| 6. | Of all nominal participants in an action denoted by a sentence, the one which is | Karta |

| | | |
|---|---|---|
| | most independent is -- | |
| 7. | ------ relations are syntactico-semantic relations between nouns and verbs. | Karaka |
| 8. | TAM stands for ---- | Tense, Aspect and Modality |
| 9. | There are --- karaka relations | Six |
| 10. | Ashraya of the result of the action is called --- | Karma |
| 11. | A verb in which ashraya of activity and result can be different, is called --- | Sakarmaka (Transitive) |
| 12. | ---- karaka is the instrument with which an action is carried out | Karana |
| 13. | ----- karaka is the locus of karta or karma | Adhikarana |
| 14. | ----- karaka is denotes separation from a stationary nominal participant | Apadana |
| 15. | ----- karaka is the beneficieary | Sampradana |
| 16. | The --- relation connects two verbs by purpose | Tadarthya |
| 17. | When two or more numbers verbs appear in a sentence, the karaka roles for each verb can be computed by --- | Karaka Sharing rules |

# 4. (A) PANINI PARSER

## Structure

## 6.0. Objectives

This chapter introduces the computation model of a language analyzer based on Paninian model. It also introduces the concepts of word sense disambiguation, and Indian traditional ways to resolve the ambiguity.

## 6.1. Introduction

We know parser is a program which extracts the information encoded in the given sentence at word, phrase and sentence level. The parser built on the basis of Paninian perspective is known as Paninian Parser. The beauty of the panini framework is the algorithmic approach which suits NLP goals i.e., extracting meaning from an utterance.

Ambiguity is having more than one interpretation. Word sense disambiguation is the process of removing ambiguity. In this section we will study how word sense disambiguation is handled in our traditional shastras.

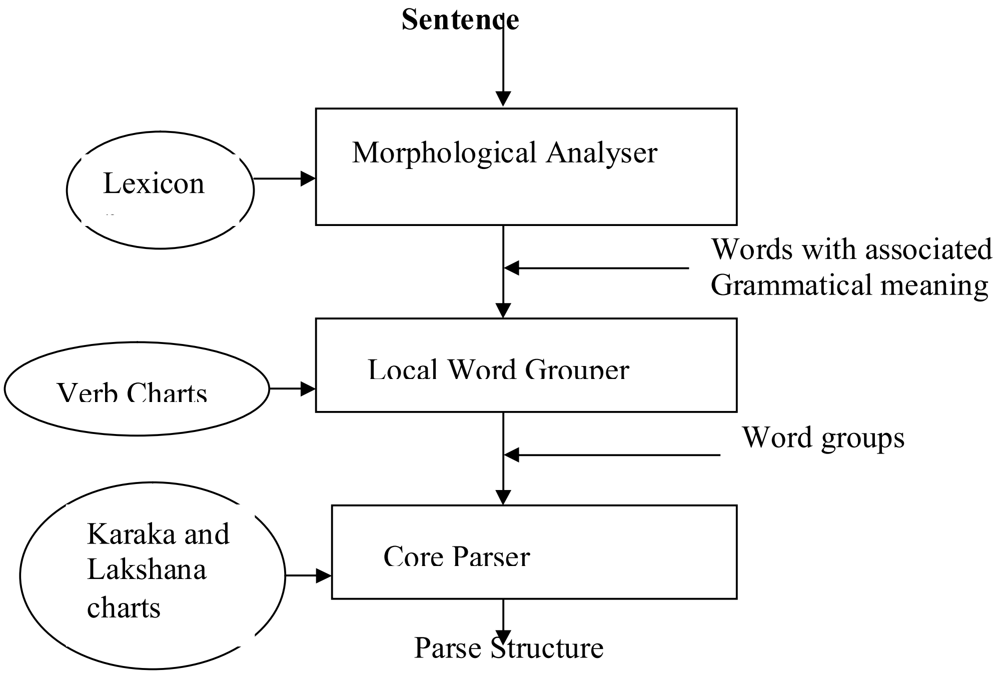
## 6.2. Panini Parser

.

The pictorial model of Paninian Parser is shown below. The parser contains the modules namely, Morphological analyzer, local word grouper and Core parser. They are describe briefly in the following sub sections, Morphological analyzer: It takes a sentence as its input. Each word of the sentence is associated with a structure indicating the root of the word, its lexical category, gender, number, person, tense ( for verbs only) etc. To achieve this a dictionary or lexicon needs to be looked up to retrieve the above information. Some times analysis of the word is also required.

For example, assume that the input sentence:

| | | |
|---|---|---|
| rAmaH | rAvanaM | mArayawi |
| Rama | Ravana | Kills. |



The output of the morphological analyzer will look like this:

rAmaH :

| | |
|---|---|
| Root | : rAma |
| Lexical Category | : noun |
| Gender | : male |
| Number | : singular |
| Person | : third |
| Case | : prathama |

rAvanaM:

| | |
|---|---|
| Root | : rAvana |
| Lexical Category | : noun |
| Gender | : male |
| Number | : singular |
| Person | : third |
| Case | : dwitiya |

mArayawi:

| | |
|---|---|
| Root | : mq |
| Lexical Category | : verb |
| Number | : singular |
| Person | : third |
| Tense | : present |

Local word grouping: The function of this block is to form word groups on the basis of the local information. For example,
English:

Look after, stand up, in front of, in stead of, etc.

Core Parser: The main function of the core parser is

1. To identify the sense of words
2. to identify karaka relations among words of the sentence

The right word sense is identified by Lakshan charts, if it has more than one meaning.. Karaka relations among words can be found from a data structure called karaka chart.

**SAQ**
Can you write a sentence in your mother tongue in nature which is ambiguous in nature.

**6.3. Karaka identification**

Karaka relations are found on the basis of Paninian theory. According to this, verb groups are called **demand groups** as they make demands about their karakas and noun groups are called **source groups** as they satisfy their demands.

For example, consider the sentence,
(RAm ne wAlA kOla)

Karaka chart for kOlA is

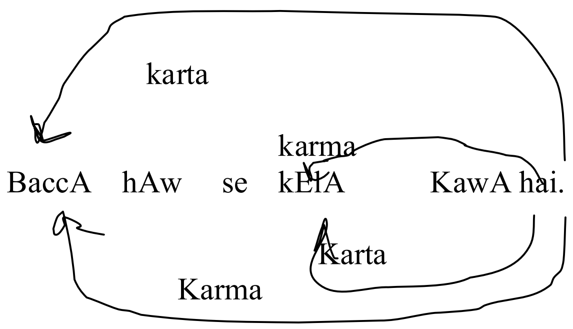
| Karaka | Vibhakti | Presence |
|---|---|---|
| Karta | ne | Mandatory |
| Karma | Null | Mandatory |
| Karana | se | optional |

The constraint graph for the above sentence is

$$\begin{tikzcd}
\text{RAm} & \text{ne} & \text{wAlA} & \text{kola} \arrow[l, bend right, "\text{karma}"'] \arrow[lll, bend left, "\text{Karta}"]
\end{tikzcd}$$

(no ambiguity)

**SAQ**
Can you write a karaka chart for a verb of your mother tongue

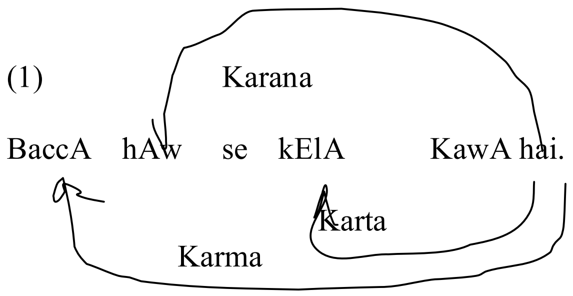


Karaka chart for KawA:

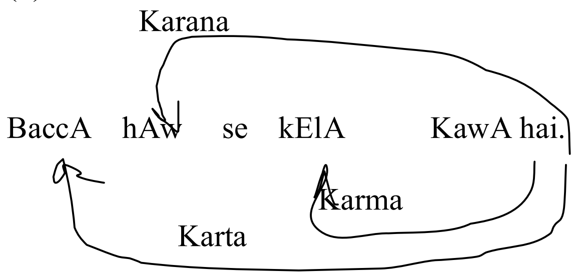
| Karaka | Vibhakti | Presence |
|---|---|---|
| Karta | Null | Mandatory |
| Karma | Null or kO | Mandatory |
| Karana | Se or dvAra | optional |

In the above sentence, both Karta and karma are tagged by null vibhakti. In such case, the finding difference between karta and karma by machine is difficult. All possible constraint graphs are drawn as shown below:

(1)



(2)



The above sub graphs are obtained using the following rules:

a) For each of the mandatory karakas in a karaka chart, for each demand group, there should be exactly one outgoing edge labeled by karaka from the demand group.

b) For each of the optional karakas in a karaka chart for each demand group, these should be at most one outgoing edge labeled by the karaka from the demand group.
c) There should be exactly one incoming arc into the each source group.

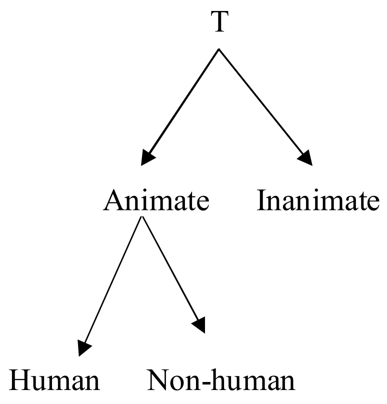
This can be done by integer programming or reducing to bipartite graph matching.

## 6.4. Word Sense Disambiguation in Panini Parser

A. Word sense may be disambiguated by using
1**)** ***Preferential Semantics***: Word knowledge helps us to disambiguate in the case of ambiguity of senses.
For Example:

baccA hAw se kelA Kawe hai

As there is no vibhakti attached to both baccA and kElA, identification of karta and karma may be difficult without additional knowledge. In such case, semantic type hierarchy shown below is used with the assumption that karta has preference in descending order.



Apart from the above, other preferential rules are also used.
1) Karta occurs before Karma
2) A source group is close to the demand group

**6.5. Lakshan charts**: Lakshan chart for verb helps us to identify the sense of a verb in the sentence. Lakshan charts make use of karakas of the verb for determining the sense.

Ex.

KisAn Kew kO jOTAwA hai.

KisAn GODO kO jOTAwA hai.

KisAn Kew kO kATwA hai.

**Here jOTAwA has two different meanings. To select the appropriate meaning of the jOTAwA, Karma is checked whether it is equivalent to the land or card. Depending on the result of checking proper sense is considered.**

**Thus noun Lakshan charts help to disambiguate the senses of noun.**

6.6. Summary

In this section, you are introduced to

- Paninian parser
- How to identify karaka and the use of karaka charts
- How to handle ambiguity with preferential semantics
- How to resolve the right sense using Lakshan charts

6.7. Model Questions

**Fifteen Marks Questions**

1) Describe the structure of Paninian Parser.

**Ten marks Questions**

1. Write a short note on Core Parsers.
2. Explain how Word Sense Disambiguation can be done by Paninian Parser.

**Five marks Questions**

1. Write a short note on Preferential Semantics.
2. Write a short note on Lakshan Charts

**One marks Questions**

| | | |
|---|---|---|
| 1 | ----- gives the mapping of verbal TAMS to karakas | Karaka Chart |
| 2 | --- make use of karakas of the verb for determining its sense | L:akshana Charts |
| 3 | -------- are called **demand groups** as they make demands about their karakas. | verb groups |
| 4 | ---------- are called **source groups** as they satisfy their demands | noun groups |
| 5. | ----------- is defined as having more than one interpretation | Ambiguity |
| 6 | ---------- deals with the identification of right sens | Word sense disambiguation |

# 4. (B) Machine Translation

## Structure

### 7.0. Objectives

In this chapter, we will learn an application of Natural Language Processing, called Machine Translation. We will be introduced to various concepts of machine Translation. We will also learn a new machine translation developed for Indian Languages, called anusaraka.

### 7.1 .Introduction

Machine Translation (MT) is the non numeric application of computers to the translation of texts from one Natural Language to another Natural language called target Language. If text in one language is given, we can get the semantically equivalent text in the languages we want. The language from which the text is converted is called source language. The language to which the text is converted is called target language.
The production of equivalent texts in a language taking natural language texts in other languages is not an easy task. Machine Translation is very difficult due to the following reasons.

For any MT System, a bilingual dictionary is required. However there is no one to one correspondence in any language pair. Therefore selection of the appropriate word in the target is problem due to the following reasons:

a) Polysemy –word having multiple meanings
b) Different shades of the meaning depending on context
c) Words belonging more than one category
d) Ambiguity due to referent pronouns
e) Lack of understanding of the text, which requires commonsense
f) Due to syntactic ambiguity of the sentence

MT is possible for a restricted domain, like office communication, railway announcements etc. However nowadays many newer techniques are making MT possible for unrestricted domain.

The brief history of Machine Translation is given below.

The research on MT started in late 1940's and many groups worked for MT in US & USSR and many primitive systems were developed but unfortunately the MT research had come down, after the release of ALPAC report. Major funds were withdrawn. However the MT field revived after late 70's. Many useful systems like TAUM-METEO were developed. From 70s onwards MT research attained full heights.

India too has active groups in IIIT, NCST, IIT Kanupur, AU-KBC, C-Dac etc.

## 7.2. Approaches used in Machine Translation

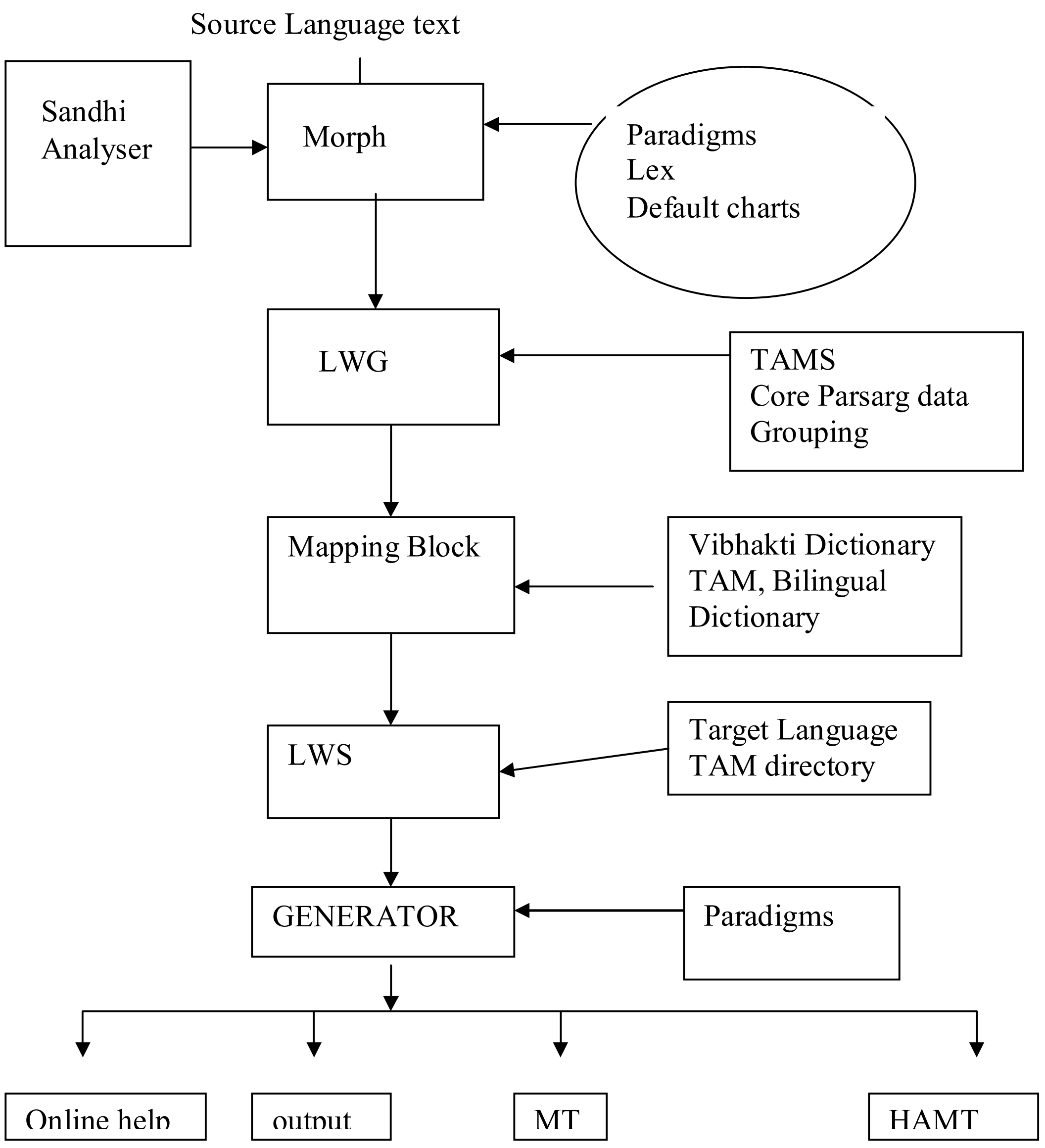
There are three possible approaches for developing MT Systems.

a) Direct approach: In this approach, MTS are developed and designed for a particular pair of languages. Therefore the systems mainly depend on the languages and cannot be used for other purposes.
b) Inter lingua approach: Here the texts in source languages are fully analysed and represented in intermediate language, called interlingua which is typically a formal mathematical language. The major advantage of this is amount of work reduced to build translation systems of large capacity. The disadvantage is that all natural language expressions must be accommodated.
c) Transfer approach: In this language, intermediate representations are unique to each language. So source language text is converted to SL representation, which in turn, is converted to TL representation. Generator acts on this TL representation and produce TL texts.

## 7.3. AnuSaaraka

Anussaraka is built to overcome the language barrier in India. Now there are five translation systems are available. I.e, from Telugu, Marathi, Kannada, Malayalam, Punjabi to Hindi respectively.

The structure anusaaraka is given below.



Morphological analyzer analyses the SL texts. If words are compound, Sandhi Analyser separates the words. It uses word paradigm table, Dictionary of indeclinable etc.

LWG groups the word sequences as a unit to give meaningful interpretation. It uses already defined grouping rules.

Now the words are mapped to TL texts by mapping block. Local word splitter (LWS) exactly behaves as the complement of LWG.

Generator produces the TL words using paradigm tables.

The output of the generator may be complex to understand. A human being may act as an editor to make this Translation System as Human Aided Machine Translation (HAMT).

**Advantages of anusaaraka:**

1. objectivity in translation
2. Forces a detailed analysis of language
3. unnecessary information need not be processed

**Disadvantages of anusaaraka** :

1. Not tuned to show the proper agreement between nouns and verbs
2. special notation make MT output difficult to interpret
3. case endings do not match.

## 7.4. Summary

In this section, we have learnt about

- What is machine Translation and why it is hard to develop Machine Translation Systems (MTS)
- General approaches used in the development of MT
- the Machine Translation system based on Paninian framework viz., anusaaraka
- advantages and disadvantages of anusaaraka

## 7.5. Model Questions

**15 Marks questions**

1.Write a Short note on Machine Translation.
2. Give brief structure of AnusAraka System.

**10 Mark questions**

1. Give possible approaches to Machine Translation system.
2. Write a short note on Anusaraka.
3. What are the advantages and disadvantages of anusaraka system.

**5 marks questions**

1. Write a short note on Preferential Semantics.
2. Write a short note on Lakshan Charts
3. What is HAMT?

**1 mark questions**

| | | |
|---|---|---|
| 1. | The process of translation of text from one language to another by using computer is called --- | Machine Translation |
| 2. | ----- Report rang the death knell of MT research in US in 1960's | ALPAC |
| 3. | TAUM-METEO in Canada is a MT from --- to -- | English French |
| 4. | ---- translates Canadian whether reports from English to French | TAUM-METEO |
| 5. | --- is MT from English to French for Textile Technology | Titus |
| 6. | --- is a MT from Chinese to English for Mathematical and Physics journals | CULT |
| 7. | --- is a MT between English and Japanese | Mu |
| 8. | There are – possible approaches for MT | 3 |
| 9. | --- approach is meant for translation of text directly from one language to another | Direct |
| 10. | In --- approach, the source language is converted into intermediate representation | Interlingua |
| 11. | Anusaraka is otherwise called as --- | Language Accessor |
| 12. | FGH-MT stands for --- | Fully-automatic General purpose high-quality |
| 13. | HAMT stands for --- | Human aided Machine Translation system |
| 14. | Morph means --- | Morphological analsyer |
| 15. | One disadvantage with anusaaraka is ----- | Readability of the output |

# 5. (A) Introduction to Perl

## Structure

---

### 8.0. Objectives

This section introduces a brief introduction to Perl. In this section, we will learn about features of Perl programming language. We will be introduced to a simple Perl program. We will also learn how to enter a Perl program into the computer and how to execute it to get the output.

### 8.1. Introduction

Perl is the most powerful, easy to use, and full featured programming language available today. A linguist, named Larry Wall, wrote Perl. Perl is an acronym for **Practical Extraction Report Language**. Perl was developed to generate reports that tracked errors and corrections to a software development project that involved multiple types of machines.

Perl, like Java Programming language, will run on a Unix, DOS, or Macintosh operating system, with little or no change required to the code we have written. This feature is called *Por tability.* The java feature of "Write once, run any where" also applies to Perl.
It is basically an interpreter i.e., it compiles and executes each line as and when it encounters in the control flow, unlike a compiler. A compiler first tries to find out the correctness of the syntax of all sentences in a program. Perl has the features of C compiler and some commands of unix like sed, awk and shell. Hence it is very much useful in manipulating texts, files and processes as well as system administration and server based tasks. There are

different versions of the Perl. The most popular versions are Perl 4 and Perl 5.

## 8.2. Perl features

The features of Perl programming language are listed below.

- Fast execution of scripts
- Integer and floating point arithmetic
- Associative arrays
- Powerful regular expression handling
- Portable across many different platforms
- Built-in system calls and shell utilities

## 8.3. Sample Perl program

Consider a sample Perl program written for Linux environment, which prints a welcome message ;

```
#!/usr/bin/perl

#The following line prints a message
print "Welcome to Rashtriya Sanskrit Vidyapeetha" ;
```

When run on any operating system, this program will produce the output

**Welcome to Rashtriya Sanskrit Vidyapeetha**

**SAQ**

Can you imitate the above sample program to print your name?

## 8.4. Execution of Perl program

Executing a program written in Perl involves a series of steps. They are given below

1. Creating the program.
2. Executing the program.

Although these steps remain the same irrespective of the operating system, system commands for implementing the steps and conventions for naming files may differ from system to system.

An operating system is a program that controls the entire operation of computer system. All input/output operations are channeled through the operating system. The operating system, which is an interface between the hardware and the user, handles the execution of user programs.

The two most popular operating systems today are UNIX (for minicomputers) and MS-DOS (for microcomputers). Now we will see how Perl programs are executed in MS DOS and Linux environment.

**Linux environment**

In Linux environment, each task can be done by issuing appropriate commands.

| Sl. No. | Activity | MS-DOS | Linux |
|---|---|---|---|
| 1 | Creating the program | Edit add.pl | vi add.pl |
| 2 | Executing the program. | perl add.pl | perl add.pl |

**SAQ**

Can you write the commands to create and run a file called "ex1.pl" on windows and Linux Platforms?

## 8.5. Summary

In this section, we had  a overall view of

- Perl Programming Language
- Features of Perl Programmming
- Creating and executing Perl programmes in different environments

## 8.6. Model Questions

### 10 Mark questions

1. How do you rune Perl programs in windows and Linux environment

### 5 marks questions

1. List the features of Perl

### 1 mark questions

| | | |
|---|---|---|
| 1. | Perl is an acronym for | Practical Extraction Report Language. |
| 2. | A Perl statement must end with a --- | Semicolon |
| 3. | The --- function outputs some information | `print` |
| 4. | "Write once, run any where" is a feature of -- and --- | `Java and Perl` |
| 5. | The feature of running a code on any operating system is called --- | `Portablility` |
| 6. | File creation command in windows is ---- | `Edit filename` |
| 7. | File creation command Linux is ---- | `vi  filename` |
| 8. | Perl execution command in window is ---- | `perl filename` |
| 9. | Perl execution command in Linux is ---- | `Perl  filename` |
| 10. | Comments can be inserted into a program with the -- symbol | # |
| 11. | Is Perl an interpreter or compiler | Interpreter |

# 5. (B) Perl Language Fundamentals

## Structure

---

### 9.0. Objectives

Whenever we learn a language we start with an alphabet. We try to construct words and use them in a sentence which conveys a basic idea. Similarly, we will study about how to construct the basic elements of Perl language, in this section. We will also learn how to name variables and how to store values in these variables. We will also learn basic operators of Perl.

### 9.1. Introduction

All computer languages can be defined in terms its syntax and semantics. Syntax is concerned with the definition of sentence structure and semantics deals with the meaning implied by these syntactic structures. Thus the task of learning a programming language is concerned with how to write syntactically correct statements to get the intended activity to be executed by the program.

In order to learn Perl language, we should learn

- to form words out of these characters
- how many different types of words are there
- how to form statements and thus form sentences.

Hence, let us try to proceed with the basic elements in the character set of Perl in this section.

## 9.2. Perl Character Set

Normally, learning Perl programming language do not begin with learning of character set. Nowhere the character set of Perl is defined and every character on the key board is a character of Perl character set. Since we are learning Perl as our first programming language, let us introduce the keys on the key board. They are

1. Letters
2. Digits
3. Special characters
4. White spaces

The compiler ignores white spaces unless they are a part of a string constant. White spaces may be used to separate words, but are prohibited between these characters of keywords and identifiers.

**Letters**

Uppercase A…..Z

Lowercase a…..z

**Digits**

Uppercase A…..Z

All decimal digits 0 …..9

**Special characters**

| | |
|---|---|
| , comma | &ampersand |
| . period | ^ caret |
| ; semicolon | *asterisk |
| : colon | - minus sign |
| ? question mark | +plus sign |

| | |
|---|---|
| ‘ apostrophe | < less than sign |
| “ quotation mark | >greater than sign |
| !exclamation mark | ( left parenthesis |
| \| vertical bar | ) right parenthesis |
| / slash | [left bracket |
| \ backslash | ]right bracket |
| ~ tilde | {left brace |
| _ under score | }right brace |
| $ dollar sign | % per cent sign |
| # number sign | |

**White Spaces and corresponding representation in Perl**

Blank space - \b

Tab -\t

Carriage return -\r

New line -\n

Form feed -\f

## 9.3. Variables

Now we will see how to form words using these characters. Words are nothing but quantities represented by variables and constants. Constants are quantities which do not change with the execution of program. Variables are quantities with change with the execution of program.

### 9.3.1. Types of variables

There are three types of variables: Scalar, array, and hash. Perl scalar variables also can hold a reference value, which is similar to a C/C++ pointer. The variable can be defined as memory location which stores some value. A constant is a value of either numeric or string type.

Variable Names are preceded by a variable designator character to indicate their type. Perl uses the first character of a variable name to distinguish which name space to allocate to a particular variable name, as follows:

- **Scalar** variables begin with the “dollar” sign ($) character
- **Arrays** start with the “at” sign (@)
- **Hashes** start with the “percent” sign (%)

Now let us learn about scalar variables in detail.

Perl scalar variable is a variable which can hold one single string or number value. Array variables store a list of values. Hash variables store a list of value pairs.

The following are the rules to be followed to name variables.

- At least one character must follow the variable designator i.e, $ sign character for scalar variables, @ for array variables and % for hash variable.
- Variable names may contain any printable character, except the space character.
- Perl is a case-sensitive language, which means that it will treat upper case letters and lower case letters as different. E.g. $name is different from $Name or $NAME etc.
- Variable names may be of any length, although some implementations may limit the variable name length to 255 characters

**SAQ**

Write the scalar variable names to represent the following quantities?

a. Temperature -------------------

b. student name ---------------------

c. the key pressed ------------

### 9.3.2. Reading scalar Variables

We can assign values (numeric and string) values to the scalar variables in two ways. They are (i) using standard input device (ii) using assignment Now we will see how a scalar variable can be assigned a value.

**(i) using standard input device (<STDIN>)**
Both numeric string values can be read using STDIN. Perl reads an entire line of input till it finds a newline each time it encounters <STDIN> in a program. If there is no input available the perl program at the standard input device, it will wait till the data is entered. It will also stop trying to read once it sees an End of File marker. Suppose you want to read a value into a variable "a",

```
print "Enter a value for a \n";
$a= <STDIN>;
```

The <STDIN> input operator returns an entire line including the newline character. Most of the times, we do not want this new line as part of out data. Hence we want to remove this new line and we can do this by

```
chop($a)
```

or simply

```
print "Enter a \n";
chop($a= <STDIN>);
```

Depending on the type of value (numeric or string), $a will be considered as numeric or string.

**SAQ**

How to read a variable called temperature

**(ii) using Assignment Operator (=)**

Using assignment operation, we can straight away assign values to the variables using an assignment operator as shown in the examples.

```
$a = 10;
```

**SAQ**

Assign 20 to a variable called temperature

Assign "Rama" to a variable called name

### 9.3.3. Printing scalar Variables

Variables can be printed using print statement. For example, if want to print a variable called $a, then consider the following statement.

```
print $a;
```

The above statement will print the contents of the variable $a.

**SAQ**
Write a print statement to print the variable $temperature

Variables also can be printed along with some literal messages. For example,

print the “Value of a is %a\n”

If $a is 5, the output of this statement will be

Value of a is 5.

**SAQ**
Write a print statement to print the variable $temperature along with the message like “The temperature of today is ----------“

### 9.4. Constants

As already told, there are two types of scalar data (1) string type which in turn is of two types- single quoted and double quoted strings (2) numeric data type

#### 9.4.1. String Constants

String is a sequence of characters which can be 8-bit ASCII value. As discussed earlier, string literals of two types viz., (1) Single-quoted strings - which are enclosed in single quotes in which no interpolation (substitution) possible; however, we can include escape sequences – which are listed below with examples (2) double quoted strings – sequence of characters enclosed in double quotes where interpolation possible.

**Single-quoted strings – Examples**

| **String** | **Its value** |
|---|---|
| ‘Sanskrit University’ | Sanskrit University |
| ‘Rama\’s book’ | Rama’s book |
| ‘’ | Null value |
| ‘a/\b’ | a/b |
| ‘she said\”’ | she said” |

**Double-quoted strings examples**

| String | Its value |
|---|---|
| "Sanskrit University" | Sanskrit University |
| "My name is $name" | My name is sree<br>Note:assume that $name contains sree |

**Escape sequences**

| | |
|---|---|
| \n | Newline |
| \r | Carriage Return |
| \t | Tab |
| \f | Formfeed |
| \b | Backspace |
| \v | Vertical Tab |
| \a | Bell |
| \e | Escape |
| \001 | Octal ASCII value (here Ctrl-A) |
| \x20 | Hex ASCII value (here space) |
| \cD | Control character (here Ctrl-D) |
| \\ | Backslash |
| \" | Double Quote |
| \l | Lowercase next letter |
| \L | Lowercase all following letters until \E |
| \u | Uppercase next letter |
| \U | Uppercase all following letters until \E |
| \E | Terminate \L or \U |

**SAQ**
Write some string constants with the examples.

**9.4.2. Numeric Constants**

Numbers are of different types. They are given below.

(1) Integers Ex: 4, -500
(2) Float Ex: 4.5, -8.3
(3) Octal Ex: O74, -O56
(4) Hexa Decimal Ex: OxAB4

## 9.5. Operators

Operators take some constants and manipulate them to produce some value. In Perl, operators can be classified as numeric and string operators.

### 9.5.1. Numeric Operations

The different types of numeric operators are listed below.

- Arithmetic : +, -, *, /, **, %

- Bitwise : &, |, ^, ~

- Comparison : ==, !=, <, >, <=, >=

For example,

| | | |
|---|---|---|
| (4 + 5) | gives | 9 |
| 9 / 4 | gives | 2.25 |
| 9.2 % 4.7 | gives | 1 |
| (4 != 4) | gives | false |

Summary with examples on Operations and Assignment

| | |
|---|---|
| $a = 1 + 2; | Add 1 and 2 and store the result 3 in $a |
| $a = 3 - 4; | Subtract 4 from 3 and store the result i.e., -1 in $a |
| $a = 5 * 6; | Multiply 5 and 6 and store in $a |
| $a = 7 / 8; | Divide 7 by 8 to give 0.875 and store it in $a |
| $a = 9 ** 10; | Compute Nine to the power of 10 and store it in $a |
| $a = 5 % 2; | Compute Remainder of 5 divided by 2 i.e., 1 and store it in $a |
| ++$a; | increment $a and then return it |
| $a++; | Return $a and then increment it |
| --$a; | Decrement $a and then return it |
| $a--; | Return $a and then decrement it |
| $a = $b . $c; | Concatenate $b and $c and store it in $a |
| $a = $b x $c; | $b repeated $c times is stored in $a |
| $a = $b; | Assign $b to $a |
| $a += $b; | Add $b to $a and store the result in $a |
| $a -= $b; | Subtract $b from $a and store the result in $a |
| $a .= $b; | Append $b onto $a and store the result in $a |

### 9.5.2. String operations

The following are some of string operators.
**(i) Concatenation: .**

Ex. "Sanskirt "."University" results in Sanskrit University

**(ii) Replication : x**

Ex. "oM" x 4 results in oMoMoMoMoM

**(iii)Comparison : eq, ne, lt, gt, le, ge**

Ex. ("Rama" lt "Sita") returns true

## 9.6. Summary

In this section, we have learnt about

- how to define, print and manipulate scalar variables

- various types of constants

- various types of operators and their functionality

## 9.7. Model questions

**10 Mark questions**
1. Write a short note on Strings in Perl (Write about single quoted, Double Quoted, escape sequences, String Operators
2. Write a short note on arithmetic operators
3) Write the Perl statements for the following
   a. To join two strings namely $firstvar and $secondvar
   b. to increment $i and store the result in $j
   c. To store the product of $i and $j in $k
   d. To read a scalar variable $name
   e. To remove a last character of the given string

**5 marks questions**
1. Write a short note on escape sequences
2. Write a program that assigns values to $firstVar and $secondVar and uses the >= operator to test their relationship to each other. Print the resulting value

**1 mark questions**

| | |
|---|---|
| The --- function outputs content of a variable or string constants | `print` |
| Scalar variables begin with -- | `$` |
| Lists or array variable begin with -- | `@` |
| Associative arrays or Hashes start with -- | `%` |
| $name is a --- variable | `Scalar` |
| @name is a --- variable | `Array` |
| %name is a --- variable | `Hash` |
| No variable interpolation is allowed in ---- | `Single quoted variable` |
| Variable names are ----- sensitive | `Case` |
| Auto increment operator is --- | `++` |
| Auto increment operator is --- | `--` |
| Substitution of a scalar variable reference with its value done inside a double-quoted string literal is called -- | `Interpolation` |
| --- Removes the last character from a scalar variable | `Chop` |
| ---Used to remove the newline from input line | `Chop` |
| --- Returns the number of characters in a scalar variable | `Length` |
| Escape character for new line in Perl is --- | \n |
| Escape character for tab in Perl is --- | \t |
| Escape character for form feed in Perl is --- | \f |
| Escape character for blank space in Perl is --- | \b |
| '.' is a string ---- operator | Concatenation |
| -- is a string repetition operator | X |
| $a +=3 is equal to --- if $a = 2 | 5 |

# 5. (C) Control Statements

**Structure**

---

## 10.0. Objectives

In this section we are learning more complex operation on numbers and string called relational expressions. When two or more relations are combined together, they are called logical expressions. Now we learn how to construct relational and logical expressions and use them in writing Perl conditional statements.

## 10.1. Introduction

When A Perl interpreter is executing the Perl program, it executes each instruction one after the other. Sometimes we may require to avoid the execution of certain instructions and sometimes the same instruction may be repeatedly executed till a condition is satisfied. In such situations we can use the control statements. Before learning control statements, let us concentrate the basic constituents of any control statements.

## 10.1. Relational operators

We often compare two compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators. We have already used the symbol '<', meaning 'less than'. Expression such as

a < b or 1 < 20

An expression containing two quantities embedded with a relational operator is termed as a relational expression. The value of a relational expression is either one or zero. It is one if the specified relation is true and zero if the relation is false. For example

10 < 20 is true

but

20 < 10 is false.

### 10.2.1. Numeric Relational Operators

The numeric relational operators are listed below. Assume that $var is 4.

| Operator | Functionality | Example | Value |
|---|---|---|---|
| > | Numeric greater than | If ($var > 9) | True |
| >= | Numeric greater than or equal | If ($var >= 3) | False |
| < | Numeric less than | If ($var < 8) | False |
| <= | Numeric less than or equal | If ($var <= 1) | True |
| == | Numeric equality | If ($var == 8) | False |
| != | Numeric non-equality | If ($var != 8) | True |

### 10.2.2. String Relational Operators

The string relational operators are listed below. Assume that $var is "rama"

| Operator | Functionality | Example | Value |
|---|---|---|---|
| .gt. | Numeric greater than | If ($var .eq. "sree") | False |
| .ge. | Numeric greater than or equal | If ($var .ge. "sree" | False |
| .lt. | Numeric less than | If ($var .lt. "sree") | True |
| .le. | Numeric less than or equal | If ($var .le. "sree") | True |

| | | | |
|---|---|---|---|
| .eq. | Numeric equality | If ($var .eq. “sree”) | False |
| .ne. | Numeric Non-equality | If ($var .ne. “sree”) | True |

**SAQ**
Write relational expressions for the following
a) average greater than 60 ------------------------------------------------

b) name not equal to “rsvp” ------------------------------------------------

## 10.3. Logical Operators

In addition to the relational operators, Perl has the following three logical operators.

| | | |
|---|---|---|
| && | meaning | logical AND |
| \|\| | meaning | logical OR |
| ! | meaning | logical NOT |

The logical operators && and || are used when we want to test more than one condition to make decisions. An example is:

($a > $ b) && ($x = =10)

An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table shown in Table below. The logical expression given above is true only if a > b is true and x= =10 is true. If either (or both) of them are false, the expression is false.
The following table is known truth table which displays all possible inputs and corresponding outputs for the logical and or operators respectively.

| Operator1 | Operator2 | Operator1&& Operator 2 | Operator1&& Operator 2 |
|---|---|---|---|
| True | True | True | True |
| True | False | False | True |
| False | True | False | True |
| False | False | False | False |

Some examples of the usage of logical expressions are:

1. (($age > 55) && ($salary < 1000))
2. (($number < 0) || ($number > 100))

Depending on the values of variables viz., age and numbers, the value of the entire expression becomes true or false.

**SAQ**
Write logical expressions for the following

To check the marks obtained in paper-1, paper-2 and paper-3 is greater than or equal to 60

**10.4. if statement**
"If" statement is used to decide whether to do something at a special point, or to decide between two courses of action.
**10.4.1. simple if**
The general format of a simple **if** statement is

```
if (condition)
{
 statement-block;
   }
statement-n;
```

The 'statement-block' may be a single statement or a group of statements. If the condition is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-n. Remember, when the condition is true both the statement-block and the statement-n are executed in sequence.
Consider the following segment of a program.

```
if ($a>$b)
{
    print " $a  is greater than $b \n";
    }
print "The values are compared\n";
```

The program compares the value of $a with the value of $b. If $a is greater than $b, it print that $a is greater than $b. Irrespective of the value of ($a>$b), the last statement of the program segment

```
print "The values are compared\n";
```

is executed

**SAQ**
Write simple if statement for the following

If average greater than 60 print the message “passed”

### 10.4.2. if-else statement

The **if…else** statement is an extension of the simple **if** statement. The general form is

```
     if (condition)
    {
       statement-block-1
     }
     else
     {
       statement-block-2
     }
   statement-n
```

If the condition is true, then the statement-block-1 (series of statements enclosed in flower brackets), immediately following the if statement are executed; otherwise, the statement-block-2 are executed. In either case, either statement-block-1 or statement-block-2 will be executed, not both. In both the cases, the control is transferred subsequently to statement-n.
Consider the following segment of a program.

```
  if ($a>$b)
  {
      print  “$a is greater than $b \n”;
      }
   else
   {
      print “$a is  less than $b \n”;
      }
  print “The values are compared\n”;
```

The program compares the value of $a with the value of $b. If $a is greater than $b, it prints that $a is greater than $b, otherwise it prints that $a is less than $b. Irrespective of the value of ($a>$b), the last statement of the program segment

```
print "The values are compared\n";
```

is executed.

**SAQ**
Write simple if statement for the following

If average greater than 35 print the message "passed" else "failed"

## 10.5. Loops

Perl gives a variety of loops so that we can write the same segment of Perl code in many alternative ways. These alternative ways make the Perl language user-friendly. The loops offered by Perl are discussed in detail in the following sub sections.

The loop is a common element in all programming languages. Loops allow you to execute a sequence of statements repeatedly, using a control variable and a keyword like for, while etc. The value of a control variable, which is really just a counter, determines how many times a statement or a statement-block is to be executed. The value of the counter is set, activated and controlled by he keyword.

### 10.5.1. While loop

The syntax of while loop is given below.

```
while (test_expr)
{
      statement-block;
 }
```

The test expression is evaluated and, if true, the statement block is executed. This continues until the test expression is false. If the text expression is false before entering into the loop, the loop body may never be executed.

Consider a problem of finding the sum of first fifty natural numbers,

```
    $i = 1;
    while ($i <= 50)
    {
      $sum =$sum +$i;
      $i++;
    }
    print ("The sum of first 50 natural numbers is $sum\n");
```

In the program segment, the sum of first fifty numbers was found out. The algorithm is trivial and hence the program.

**SAQ**
Write while statement to print the first 10 numbers

**10.5.2. Until loop**

The syntax of until statement is given below:

```
until (test_expr)
{
    statement;
}
```

The test expression is evaluated and, if false, the statement block is executed. This continues until the test expression is true. Note that the loop body may never be executed, if the text expression is evaluated true before entering into the loop for the first time.

Ex.

```
    $i = 50;
    until ($i > 1)
    {
      $sum =$sum +$i;
    $i--;
}
print ("The sum of first 50 natural numbers is $sum\n");
```

The same problem discussed in the 5.4.2 was written using until loop. Try to trace the program to understand clearly.

**SAQ**
Write until statement to print the first 10 numbers

### 10.5.3 Do-while

The syntax of do-while is shown below.

```
do
{
        body of the loop
}
while (test_expr);
```

on reaching the do statement, the program proceeds to executethe body of the loop first. At the end of the loop, the condition in the while statement is evaluated. If the test_expr is true, the program continues to evaluate the body of the loop once again. This process continues as long as the test_expr is true. When the test_expr becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

Since the condition is evaluated at the bottom of the loop, the do…while construct provides an exit-controlled loop and therefore the body of the loop is always executed at least once.

A simple example of a do… while loop written for the problem discussed in the previous sections is given below:

```
$i = 1;
$sum=0;
 do
{
  $sum =$sum +$i;
  $i++;
}
while ($i<=50);
```

**SAQ**
Write do..while statement to print the first 10 numbers

**10.5.4. do..until**
The syntax of do.. until is shown is below.

```
do
{
    body of the loop
}
until  (condition);
```

The statement block is executed. Then the condition is evaluated and, if false, the statement block is executed. This continues until the condition is true. Note that the loop body is executed at least once.

An example of do..until statement is

```
$i = 50;
$sum=0;
 do
{
  $sum =$sum +$i;
  $i--;
}
until ($i>0);
```

**SAQ**
Write do..until statement to print the first 10 numbers

**10.5.5. For statement**

The syntax of For statement is given below.

```
for (initial_expr; test_expr; increment_expr)
  {
   Statement-block;
  }
```

The initial expression is evaluated first. Then the test expression is evaluated and, if true, the statement block s executed. Then the increment expression is evaluated and the test expression re-evaluated. This continues until the test expression is false. Note that the loop is equivalent to

```
initial_expr;
while (test_expr)
{
  Statement-block;
  increment_expr;
}
```

Example

```
$sum=0;
for ($i = 1; $i <= 50; $i++)
{
  $sum = $sum +$i;
}
print ("The sum of first 50 natural numbers is $sum\n");
```

**SAQ**
Write for statement to print the first 10 numbers

## 10.6. Summary

In this section, we have learnt about how to
-construct relational and logical operators
-how to write conditional statements using if statements
-how to write loops for execution of repeated block Perl statements using

- While statement
- Until statement
- Do..while statement
- Do..until statement

## 10.7. Model Questions

**15 Marks questions**

1. Use the while loop in a program to count from 1 to 100 in steps of 5.
2. Use the for loop in a program to print each number from 55 to 1.
3. Write a program in perl to accept the two numbers and print the odd numbers between them.
4. Write a program in Perl to print the multiplicative table of a given number.
5. Write a program in Perl to print the two digit number into its corresponding word form. (Example. Input :12 -output : Twelve)
6. Write a program in Perl to print the two digit number into its corresponding word form. (Example. Input :12 -output : One Two)
7. Write a program in Perl to print the given word into numeric form. (Example. Input :Twelve -output : 12)

**10 Mark questions**

1) Briefly explain the syntax if – else statement with an example
3) Explain while statement with an example
4) Explain until statement with an example
5) Explain do statement with an example
6) Explain do- while statement with an example
7) Explain for statement with an example
8) Explain for-each statement with an example

**5 marks questions**

1. Write a short note on comparison (relational operators) in Perl
2. Explain logical operations with appropriate examples.
3. Write a program in Perl to print your name ten times using any control loop.

**1 mark questions**

| | |
|---|---|
| String Comparison operator for "=" is --- | eq |
| String Comparison operator for "<" is --- | lt |
| String Comparison operator for "<=" is --- | le |
| String Comparison operator for "!=" is --- | ne |
| String Comparison operator for ">" is --- | gt |
| String Comparison operator for ">=" is --- | ge |
| Two way branching control statement is -- | if |
| User can read a value using ---- File handle in Perl | <STDIN> |
| ---- loop repeats a block of code as long as a condition is ture | While |

# 5. (D). Array Variables

**Structure**

---

## 11.0. Objectives

In previous sections, we learnt about the first type of variable namely, scalar variable. In this section, we will learn about the second type of variable called array variable.

## 11.1. Introduction

Scalar variable stores a single value. Sometimes, we may require to store a group of values together. In such situations, we use array variables. An array variable is a list of scalars. Remember a scalar value can be either numeric or string. Now let us see how to define an array and manipulate it.

## 11.2. Array Constants

Here are a few examples of some array constants, using two operators - () and qw operator.

(i) An empty list defined using () operator
```
();
```
(ii) An empty list defined using qw operator
```
qw//;
```

(iii) A list with three elements viz., a,b,c
qw /a b c/;
(iv) an array constant with three elements viz., a,b,c
(“a”,”b”,”c”)
(v) a list constant with six elements consisting of a, b, c, 1, 2, 3
("a", "b", "c", 1, 2, 3);
(vi) Another list consisting of six words namely - hello, world, how, are, you today
qw/ hello world how are you today /;
Finally, if you have any two scalar values where all the values between them can be enumerated, you can use an operator called the slice operator represented by .. to build a list. This is seen in an example:
(vii) a list of 100 elements: the numbers from 1 to 100
(1 .. 100);
(viii) a list of 26 elements: the uppercase letters From A to Z
('A' .. 'Z');
(ix) a list of 31 elements: all possible days of a month
('01' .. '31');

**SAQ**

1) Define the different colours as elements of a list

2) Define your marks obtained in the test as a list

**11.3. Array variables**
Array variables is a data structure which stores an array constant. The rules for naming an array variable are same as those for naming a scalar variable except for initial character which starts with an @ symbol. The statement defines and initializes an array variable with three elements – s, bananas, oranges.
@fruits = ("apples", "bananas", "oranges");
or
@fruits = qw/apples banana oranges /;
which is more easier.
Consider another example-
@music_instruments = ("whistle", "flute", “veena”, “guitar”);

Which assigns four elements - whistle, flute,veena, guitar- to the array variable @music_instruments.
The array is accessed by using indices starting from 0, and square brackets are used to specify the index. The expression $fruits[2] returns oranges. Notice that the **@** has changed to a **$** because *"oranges"* is a scalar. New members also can be add to the array in the following way.
$fruits[3] = "Pomegranate";
Similarly $music_instrumen[4]="Mrudangam";
The array can be assigned a set of values as given above. Further we can define another array using the definitions of previously declared arrays as shown below.
(i) Consider the following example
@more_music_instruments = ("mouth_organ", @music_instruments, "Talam");
which explodes @music_instruments as whistle, flute,veena, guitar resulting in
@more_music_instruments=("mouth_organ","whistle","flute","veena", "guitar","Talam");
(ii) This should suggest a way of adding elements to an array. Another way of adding elements is to use push statement as shown below
push(@more_music_instruments, "Tabala" )
which pushes "Tabala" onto the end of the array @more_music_instruments which contains the elements - whistle, flute,veena, guitar, Tabala
To push two or more items onto the array, use one of the following forms:
push(@more_music_instruments, "Tabala" , "Drums");
To remove the last item from a list and return it, we use the **pop** function. From our original list @more_music_instruments, the **pop** function will return "drum" to the scalar variable $music_instrurment5.
$music_instrurment5 = pop(@more_music_instruments \);
It is also possible to assign an array to a scalar variable. As usual context is important. The line
$no_of_instruments = $#music_instrurments;
or
$no_of_instruments = @music_instrurments;
assigns the number of elements contained in the array @music_instrurments to a scalar variable - $no_of_instruments, but
$ names_of_music_instrurment = "@music_instrurments";
turns the list into a string with a space between each element.
Arrays can also be used to make multiple assignments to scalar variables:
(i) ($a, $b) = ($c, $d);
Same as $a=$c; $b=$d;
(ii) ($a, $b) = @fruits;
$a and $b are initialized with the first two items of @fruits.

**SAQ**

1) Define different colours as elements of a list called colours

2) Add a new colour to the above list viz., colours

3) Remove an element from the list

**11.4. . Reading array variables**
We can read or initialize array variables as defined in the previous section.
(i) @music_instruments = ("whistle", "flute", “veena”, “guitar”);
(ii) Another way of reading an array is to use standard input device as given bellows;
@music_instruments = <STDIN>;
When we use standard input device for reading in the list context, the system will try to read the list of the variables until you press Control -D or Control-Z depending on the operating system. So we have to type all the elements of the list one by one and then press Control-D simultaneously.

**SAQ**

1) Define different colours as elements of a list called colours

2) Add a new colour to the above list viz., colours

3) Remove an element from the above list

## 11.5. Printing array variables

We can print the arrays in many ways.

(i) print @music_instruments;

(ii) print “The last element of music instruments is $music_instruments[5]\n”;

In the first example, the array is printed as it is. In the second example we can print any specific element as shown in the example.

### 11.5.1. Using for each statement

This statement iterates through the list of values one at a time, beginning from the first element to the last element. For example,

```
foreach $instrument (@music_instruments)
{
 print “One instrument is $instrument  \n”
}
```

Here $instrument is the control variable which takes the first element for the first iteration, next element for the next iteration and so on until all the elements in the list are exhausted. Otherwise, we can make use of special variable $_ as control variable as shown below.

```
foreach   (@music_instruments)
{
 print “One instrument is $_  \n”
}
```

**SAQ**

Write a Perl program to read and print the given array.

## 11.6. Sort and Reverse Operators

The sort operator takes an array and sorts in ascending order. Since Perl is case-sensitive, all words which start with upper case letters come first and words with lowercase letters come last. For example,

@sorted_mustic_instruments = sort (@music_instruments);

Then the array @sorted_mustic_instruments contains the elements in sorted order. For example, consider the program:

```
@names  = (“sita", “raju”, “amar”, “kavita”);
@sorted_name = sort(@names);
foreach $name (@sorted_name)
```

```
{
 print “$name  \n”
}
```

The output of the program will be -

```
amar
kavita
raju
sita
```

Similarly the reverse operator reverses the list i.e., takes the given list in opposite order. For example, consider the program-

```
@names  = (“sita", “raju”, “amar”, “kavita”);
@reverse_name = reverse(@names);
foreach $name (@reverse_name)
{
 print “$name  \n”
}
```

The output of the program will be -

```
kavita
amar
raju
sita
```

Let us see some example programs which use array variables.

Ex.1 – Write a program in Perl to read in array of your subject names and print them along with serial number.

**Solution**

```
print “Enter the names of subjects  and Press Control-D at the end \n”;
@subject_names= <STDIN>;
#removes the end of line of each array
chomp(@subject_names);
#set serial_number to zero
$serial_no=0;
foreach $name (@reverse_name)
{
 print “The serial number  of  $name $serial_no \n”;
 $serial_no++;
}
```

Ex. 2 – Write a program in Perl to read in array of numbers and print the sum of the numbers in the list.

**Solution**

```
print “Enter some numbers   and Press Control-D at the end \n”;
@numbers = <STDIN>;
#removes the end of line of each array
chomp(@numbers);
#set sum  to zero
$sum=0;
foreach $number (@numbers)
{
  $sum  += $number;       # it can also be written as $sum = $sum+$number

}
print “The sum of all numbers in the list is $sum\n”;
```

Ex. 3– Write a program in Perl to read in array of numbers and change all odd numbers into “zero” and print the resultant array.

**Solution**

```
print “Enter some numbers   and Press Control-D at the end \n”;
@numbers = <STDIN>;
#removes the end of line of each array
chomp(@numbers);
for ($i=0; $i <= $#@numbers; $i++)
{
   if ( ($numbers[$i]  mod 2) ==1)
   {
      $number[$i] = “zero”;
   }
   print “The $i th element of number array is $numbers[$i] \n”;
}
```

## 11.7. Summary

In this section, we have learn about

- Array constants
- How to define, initialize and manipulate array variables
- How to read and print array variables
- Functions than can be used with array structures
    - Foreach
    - Sort
    - reverse

## 11.8. Model Questions

### 15 Marks questions

1.Write a program in Perl to read in array of your subject names and print them in descending sorted order along with serial number.
2.Write a program in Perl to read in array of numbers and print the square of each numbers in the list
3.Write a program in Perl to read in array of names and print the index of an element if the name is given.
4.Create an array called `@months`. It should have 12 elements in it with the names of the months represented as strings. Write a Perl program to accept a given number, and print the month accordingly.
5.Create an array called `@months`. It should have 12 elements with the names of the months represented as strings. Write a perl program to accept the name of the month and print what is its cardinality in the year. ( May – 5)
6.Using the range operator (..), create an array with the following elements: 1, 5, 6, 7, 10, 11, 12. Write a Perl program to display the element if its index is given.

### 10 Mark questions

1. Write a short note on arrays.
2. Write a short note on the following functions
   a. sort b. reverse c. foreach

### 5 marks questions
1. Explain the terms (a) Push (b) Pop

### 1 mark questions

| | |
|---|---|
| List of comma-separated scalar values --- | `Array` |
| List of elements selected from an array is called --- | `Slice` |
| The length of array whose name is "students" can be found by --- | $#students |
| --- is used to make double quoted string with a short cut | qw |
| --- operator takes the last element off an array | pop |
| --- adds an element to a list | push |
| --- statement is very useful to process each every element of list and hash | foreach |

| | |
|---|---|
| When we don’t want to specify a variable name or when we refer to a default variable, then we use -- | $_ |
| ---- function reverses the contents of an array | reverse |
| ---- function sorts the contents of an array | sort |

# 5. (E) Hash (Associative Arrays) variables

**Structure**

---

## 12.0. Objectives

In previous sections, we learnt about the first and second type of variables namely, scalar and array variables. In this section, we will learn about the third type of variable called Hash variable.

## 12.1. Introduction

Ordinary list arrays allow us to access their element by number. The first element of array @food is $food[0]. The second element is $food[1], and so on. But Perl also allows us to create arrays which can be accessed by string as its index. These are called associative arrays.
The main features of Associative arrays are :

- Collection of scalar data
- Elements have no particular order
- Index values are arbitrary scalars and not necessarily integers, as is in an array
- Index values are called keys
- The actual elements referred by keys are called values

## 12.2. Defining Hashes

To define an associative array we use the usual parenthesis notation, but the array itself is prefixed by a % sign. Suppose we want to create an array of fruits and their corresponding color. It would look like this:

```
%fruits  = ("apple", “red”,
                "grapes”, “black”,
                "banana", “yellow”,
                 "orange", "green" );
```

Now we can find the colour of a given fruit with the following expressions -

```
$fruits{"apple"};                # Returns  “red”
$fruits{"grapes"};               # Returns  “black”
```

Note that **%** sign has changed to a **$** to access an individual element because that element is a scalar. Unlike list arrays, the index (in this case the fruit's name) is enclosed in curly braces, the idea being that associative arrays are fancier than list arrays.

An associative array can be converted back into a list or array just by assigning it to a list array variable. Ideally the list array will have an even number of elements:

(i) @info = %fruits;

@info is a list array. It has now 8 elements i.e., apple, red, grapes, black etc., after conversion of hash viz., fruits into an array variable.

(ii) $info[5];

returns the value “yellow”

(iii) %more_fruits = @info;

A hash variable %more_fruits is an created and it is same as %fruits

SAQ

Define a hash consisting of subjects you have taken as keys and the names of teachers as values

## 12.3. Keys Function

Keys function returns a list of the keys of an associative array when used in an array context. It also returns the number of elements (key-value pairs) when used in a scalar context. The list of keys is not returned in sorted order, but rather in an random order determined by the internal way as Perl stores it an associative array.

For example consider the following commands,

(i) @fruitnames= keys %fruits;

The list fruitnames initialized with the key elements - apple, grapes, banana and orange.

(ii) $fruitnumber = keys %fruits;

The scalar variable $fruitnumber contains the number of key elements in the %fruits. In this case, it is four.

## 12.4. Values Function

Values function gives a list of the values of an associative array when used in an array context and gives the number of elements (key-value pairs) when used in a scalar context.
For example consider the following commands,
(i) @fruitcolours= values %fruits;
The list fruitcolours is initialized with the elements red, black, yellow and green.
(ii) $fruitnumber = values %fruits;
The scalar variable $fruitnumber contains the number of key elements in the %fruits. In this case, it is four.

SAQ
Define a hash consisting of subjects you have taken as keys and the names of teachers as values. Now extract subject and teacher names into two other arrays.

## **12.4. Each Function**

Each function gives an element (key-value pair) of an associative array as a two-element list when used in an array context. Each associative array has a corresponding iterator. On every use of the each operator, the iterator advances to the next key-value pair. When the end of the associative array is reached, the empty list is returned. The next use of each stops the iteration process.
Now consider the program;

```
   %fruits  = ("apple", “red”,
                "grapes”, “black”,
                "banana", “yellow”,
                 "orange", "green" );
while (($name, $colour) = each (%fruits))
 {
        print ("The colour for $name is $colour \n");
   }
```

The program is self explanatory. The output of the program looks like :

```
The colour for apple is red
The colour for  grapes is black
The colour for  banana  is yellow
The colour for orange is  green
```

SAQ
Define a hash consisting of subjects you have taken as keys and the names of teachers as values. Now for each subject, print the name of the teacher

Delete Function

Delete function is used to delete an element (key-value pair) of an associative array and it returns the element as a two-element list. For example,

```
        delete ($fruits{"apple"});
```

deletes the key-value pair "apple" and "red" from the associative array.

SAQ
Define a hash consisting of subjects you have taken as keys and the names of teachers as values. Now delete the last subject from the hash.

**12.5. Reading hash variables**

The hash variable can be constructed in the following ways.

(i) **using assignment operator** as discussed in the previous section. For example,

```
   %fruits  = ("apple", "red",
               "grapes", "black",
               "banana", "yellow",
                "orange", "green" );
```

Creates a hash variable with the following key-value pairs.

| Key | value |
|---|---|
| apple | red |
| grapes | black |
| banana | yellow |
| orange | green |

**(ii) By reading**

To create the above hash %fruits , we can use the following Perl code. Assume that we give data - the key and its associated value line by line in the following way :

```
apple,red
grapes,black
banana,yellow
orange,green

# Give literal message how data should be entered
# To indicate the end of data proper instruction is given to the student

print “Type the fruit name and its colour separated by “\,” and press
Control_D to indicate the end of data \n”;
# since we do not know the number of fruits, we are use undening loop
while (<>)
{
            chomp;
            my ($name, $colour) = split /, /;
             $fruits{$name} = $colour;
           }
 while (($name, $colour) = each (%fruits))
 {
        print ("The colour for $name is $colour \n");
   }
```

## 12.6. Printing hash variables

We have already seen how elements are to printed using each function in the previous section.

See the following example programs

**Example-1**

Write a program in Perl to create a hash by reading a state and its capital and print the hash in sorted orders of state names:

**Solution**

```
# Give literal message how data should be entered
# To indicate the end of data proper instruction is given to the student
print “Type the state name and its capital  separated by  “\,” and press
Control_D to indicate the end of data \n”;
# since we do not  know the number of fruits, we are use unending loop
while (<>)
{
              chomp;
              my ($name, $capital) = split /, /;
               $state_capital{$name} =  $capital;
             }
@states= keys %state_capital;
foreach   $state (sort @states)
 {
        print ("The capital  for $state is $state_capital {$state}  \n");
   }
```

**Example-2**

Write a program in Perl to create a hash by reading a state and its capital and print the capital name if state is given.

**Solution**

```
 # Give literal message how data should be entered
# To indicate the end of data proper instruction is given to the student

print “Type the state name and its capital  separated by  “\,” and press
Control_D to indicate the end of data \n”;

# since we do not  know the number of fruits, we are use unending loop

while (<>)
{
              chomp;
              my ($name, $capital) = split /, /;
               $state_capital{$name} =  $capital;
             }
 #read the state name for which capital is to be printd
 chop($state =<STDIN>);
 print ("The capital  for $state is $state_capital {$state}  \n");
```

Example-2

Write a program in Perl to accept a series of words and prints how many times each word appears in the text.

**Solution**

```
# Give literal message how data should be entered
# To indicate the end of data proper instruction is given to the student

print "Type series of words line by line  and press Control_D to indicate the
end of data \n";
chop (@text=<STDIN>;
 #create a hash  taking each word as key and the number of  times it appears as
 its value
 foreach  $word (@text)
 {
  #create a hash $word_count  with word as key and the number of times it
 appear as its value
   $word_count{$word} = $word_count{$word}+1;
 }
while (($word, $count) = each (%word_count))
 {
        print ("The word $word appears $count times \n");
   }
```

## 12.8. Summary

In this section, we  have learn about

- Hash  variable
- How to define, initialize and manipulate  hash variables
- How to read and print hash variables
- Functions  than can be used with hash structures
    - delete
    - keys
    - values
    - each

## 12.9. Model Question

**15 marks questions**

1. Write a program in Perl to create a hash  by the names of all months in a year  and the number of days in that month and to print number  of days in the name of the month is given.
2. Write a program in Perl to create a dictionary of English – Telugu (or any language you know) words  using hash and to print the Telugu word if English word is given.

**10 marks questions**

1. Write a short note on hashes
2. write a short note on any two of the following

   a. Keys b. Values c. delete d. each function

**5 marks questions**

1. How do you read hashes
2. How do you print hashes
3. How do you initialize a hash

**1 marks questions**

| | |
|---|---|
| --- function yields a list of all keys in the hash | Keys |
| --- function yields a list of all values in the hash | Values |
| --- returns key-value pair of a hash | Each |
| ---- -- deletes an element from the hash | Delete |
| -------- function is associated to process each element of a hash | each |