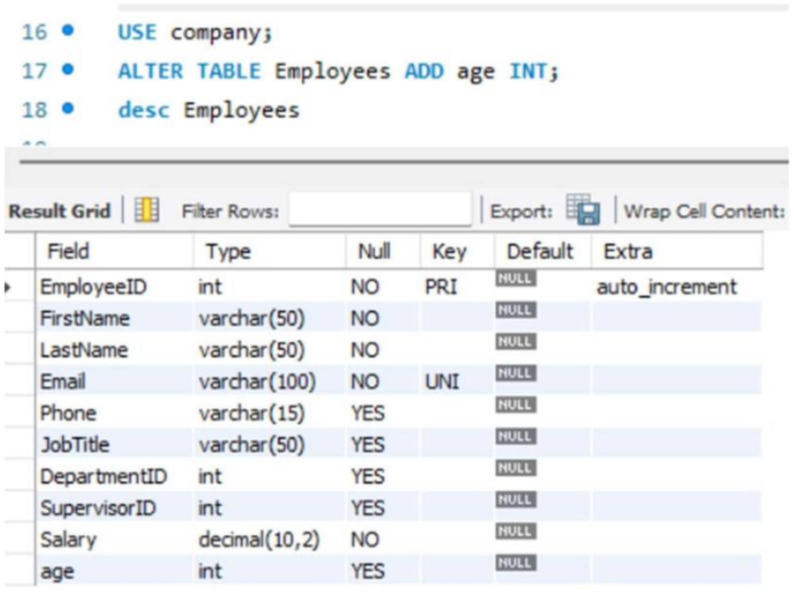
Name: Ankit raj   **Roll No.:-11233036**   **Department/Semester/Group: CSE/4/G1**

PRATICAL-4

AIM: To Describe the Structure of Tables of Company DBMS.

◉ **Add a New Column:** You can add a new column to an existing table.

```
ALTER TABLE table_name
ADD column_name datatype;
```



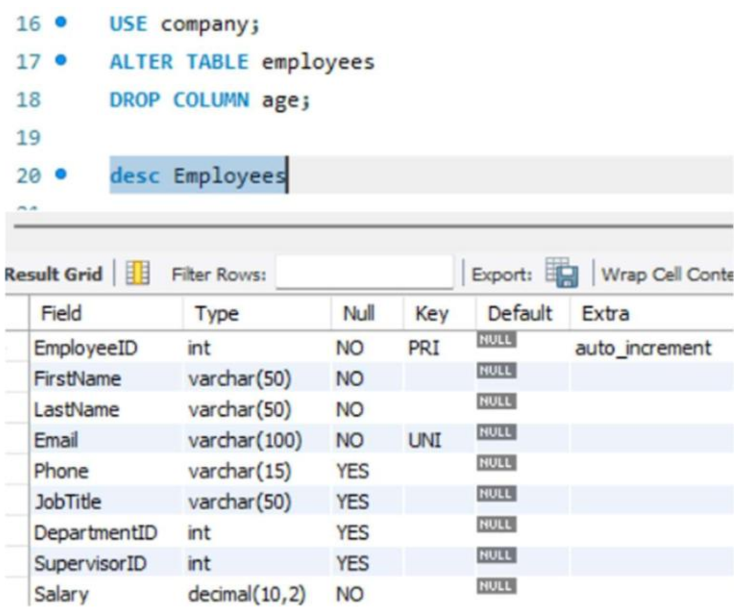
```
USE company;
ALTER TABLE Employees ADD age INT;
desc Employees
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
EmployeeID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(50)	NO		NULL	
LastName	varchar(50)	NO		NULL	
Email	varchar(100)	NO	UNI	NULL	
Phone	varchar(15)	YES		NULL	
JobTitle	varchar(50)	YES		NULL	
DepartmentID	int	YES		NULL	
SupervisorID	int	YES		NULL	
Salary	decimal(10,2)	NO		NULL	
age	int	YES		NULL	

◉ **Drop a Column :** Remove a column from the table.

```
ALTER TABLE table_name
DROP COLUMN column_name;
```



```
USE company;
ALTER TABLE employees
DROP COLUMN age;

desc Employees
```

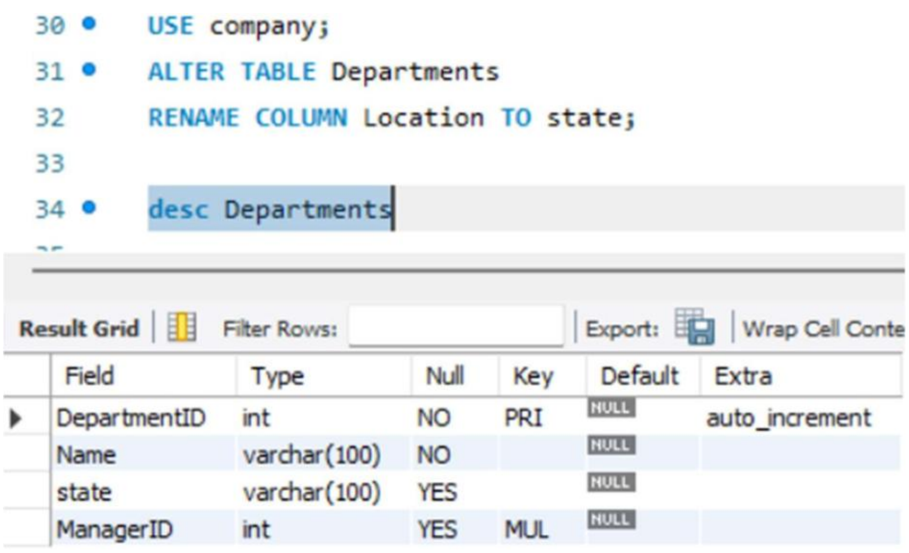
Result Grid | Filter Rows: | Export: | Wrap Cell Conte

Field	Type	Null	Key	Default	Extra
EmployeeID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(50)	NO		NULL	
LastName	varchar(50)	NO		NULL	
Email	varchar(100)	NO	UNI	NULL	
Phone	varchar(15)	YES		NULL	
JobTitle	varchar(50)	YES		NULL	
DepartmentID	int	YES		NULL	
SupervisorID	int	YES		NULL	
Salary	decimal(10,2)	NO		NULL	

◉ **Rename a Column** (Supported in some databases like MySQL, PostgreSQL, etc.)

Rename an existing column.

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name;
```



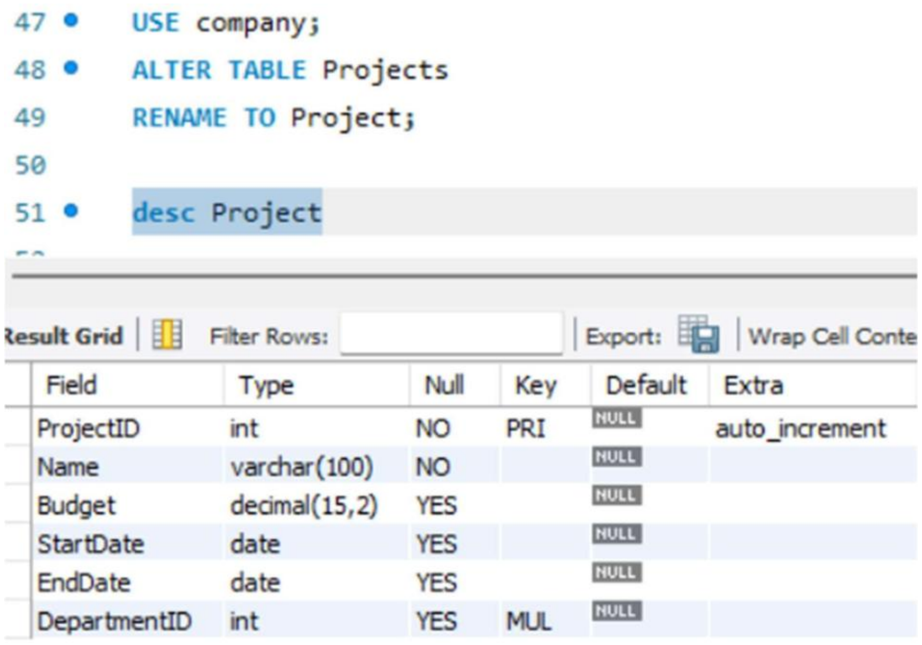
```
USE company;
ALTER TABLE Departments
RENAME COLUMN Location TO state;

desc Departments
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

Field	Type	Null	Key	Default	Extra
DepartmentID	int	NO	PRI	NULL	auto_increment
Name	varchar(100)	NO		NULL	
state	varchar(100)	YES		NULL	
ManagerID	int	YES	MUL	NULL	

◉ **Rename a Table :** Rename the entire table.

```
ALTER TABLE old_table_name
RENAME TO new_table_name;
```



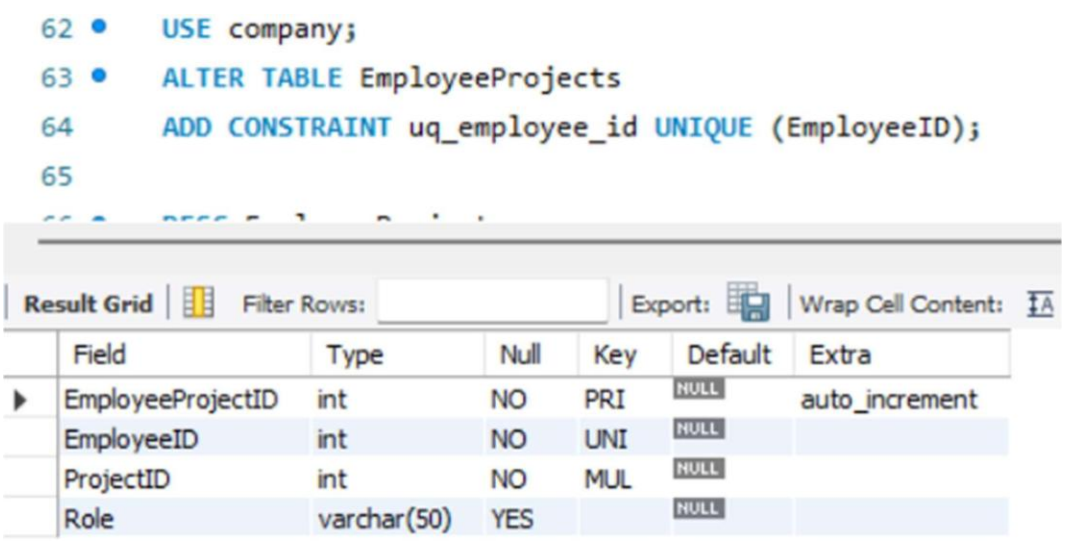
```
USE company;
ALTER TABLE Projects
RENAME TO Project;

desc Project
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

Field	Type	Null	Key	Default	Extra
ProjectID	int	NO	PRI	NULL	auto_increment
Name	varchar(100)	NO		NULL	
Budget	decimal(15,2)	YES		NULL	
StartDate	date	YES		NULL	
EndDate	date	YES		NULL	
DepartmentID	int	YES	MUL	NULL	

◉ **Add a Unique Constraint to an Existing Column:**

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE (column_name);
```

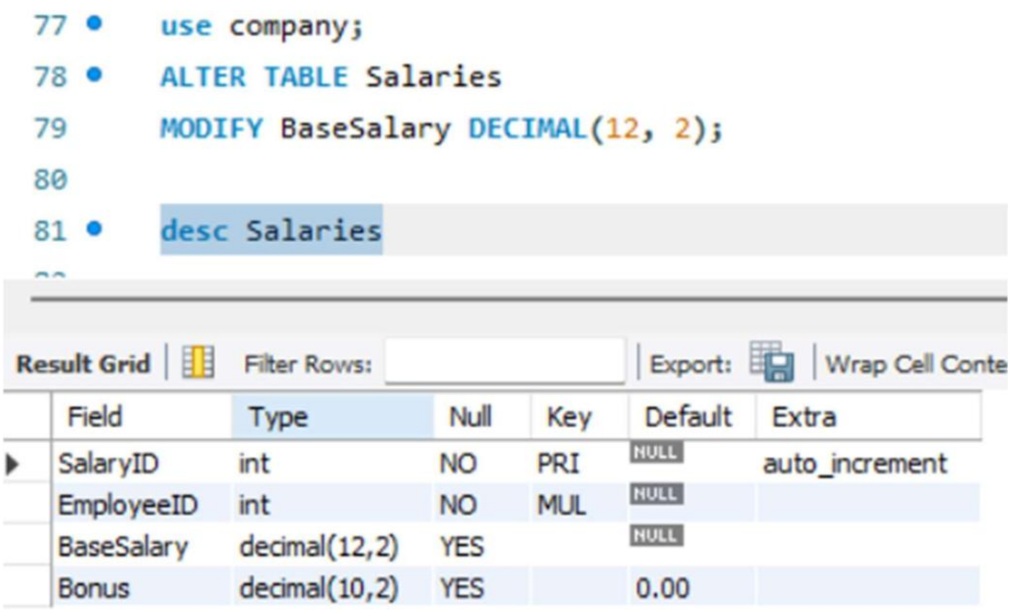


```
USE company;
ALTER TABLE EmployeeProjects
ADD CONSTRAINT uq_employee_id UNIQUE (EmployeeID);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
EmployeeProjectID	int	NO	PRI	NULL	auto_increment
EmployeeID	int	NO	UNI	NULL	
ProjectID	int	NO	MUL	NULL	
Role	varchar(50)	YES		NULL	

◉ **Modify an Existing Column to Add a Unique Constraint**



```
use company;
ALTER TABLE Salaries
MODIFY BaseSalary DECIMAL(12, 2);

desc Salaries
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

Field	Type	Null	Key	Default	Extra
SalaryID	int	NO	PRI	NULL	auto_increment
EmployeeID	int	NO	MUL	NULL	
BaseSalary	decimal(12,2)	YES		NULL	
Bonus	decimal(10,2)	YES		0.00	

Name: Ankit raj Roll No.:-11233036 Department/Semester/Group: CSE/4/G1

EXPERIMENT NO :-04

AIM:- . Solve 8-puzzle problem using Best First Search. Write a program to Implement A*

```
import heapq

def a_star_search(start, goal):
    def heuristic(state):
        distance = 0
        for i in range(len(state)):
            if state[i] != 0:
                x1, y1 = divmod(i, 3)
                x2, y2 = divmod(goal.index(state[i]), 3)
                distance += abs(x1 - x2) + abs(y1 - y2)
        return distance

    def get_neighbors(state):
        neighbors = []
        zero_index = state.index(0)
        x, y = divmod(zero_index, 3)
        moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for dx, dy in moves:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 3 and 0 <= ny < 3:
                new_index = nx * 3 + ny
                new_state = list(state)
                new_state[zero_index], new_state[new_index] = new_state[new_index],
new_state[zero_index]
                neighbors.append((tuple(new_state), zero_index, new_index))
        return neighbors

    def move_description(prev_index, new_index):
        directions = {(-1, 0): "up", (1, 0): "down", (0, -1): "left", (0, 1): "right"}
        x1, y1 = divmod(prev_index, 3)
        x2, y2 = divmod(new_index, 3)
        dx, dy = x2 - x1, y2 - y1
        return directions[(dx, dy)]

    open_set = []
    heapq.heappush(open_set, (0 + heuristic(start), 0, start, [], None))
    visited = set()
    while open_set:
        f, g, current, path, last_move = heapq.heappop(open_set)
        if current in visited:
            continue
        visited.add(current)
        path = path + [(current, last_move)]
        if current == goal:
            return path
        for neighbor, zero_index, new_index in get_neighbors(current):
            if neighbor not in visited:
                move = move_description(zero_index, new_index)
                heapq.heappush(open_set, (g + 1 + heuristic(neighbor), g + 1, neighbor, path,
move))
    return None

start_state = (1, 2, 3, 4, 5, 6, 7, 8, 0)
goal_state = (1, 2, 3, 0, 5, 6, 7, 8, 4)

solution = a_star_search(start_state, goal_state)
if solution:
    print("Solution found:")
    for step_num, (step, move) in enumerate(solution):
        print(f"Step {step_num}: {move if move else 'Start'}")
        for i in range(0, 9, 3):
            print(step[i:i+3])
        print()
else:
    print("No solution exists.")
```

OUTPUT:-

```
E:\python\python.exe "C:\Users\Ankit raj\Documents\4th sem exp\puzzle.py"

Solution found:

Step 0: Start

(1, 2, 3)

(4, 5, 6)

(7, 8, 0)


Step 1: up

(1, 2, 3)

(4, 5, 0)

(7, 8, 6)
```

Step 2: left

(1, 2, 3)

(4, 0, 5)

(7, 8, 6)

Step 3: left

(1, 2, 3)

(0, 4, 5)

(7, 8, 6)

Step 4: down

(1, 2, 3)

(7, 4, 5)

(0, 8, 6)

Step 5: right

(1, 2, 3)

(7, 4, 5)

(8, 0, 6)

Step 6: up

(1, 2, 3)

(7, 0, 5)

(8, 4, 6)

Step 7: right

(1, 2, 3)

(7, 5, 0)

(8, 4, 6)

```
Step 8: down
(1, 2, 3)
(7, 5, 6)
(8, 4, 0)

Step 9: left
(1, 2, 3)
(7, 5, 6)
(8, 0, 4)

Step 10: left
(1, 2, 3)
(7, 5, 6)
(0, 8, 4)

Step 11: up
(1, 2, 3)
(0, 5, 6)
(7, 8, 4)
Process finished with exit code 0
```

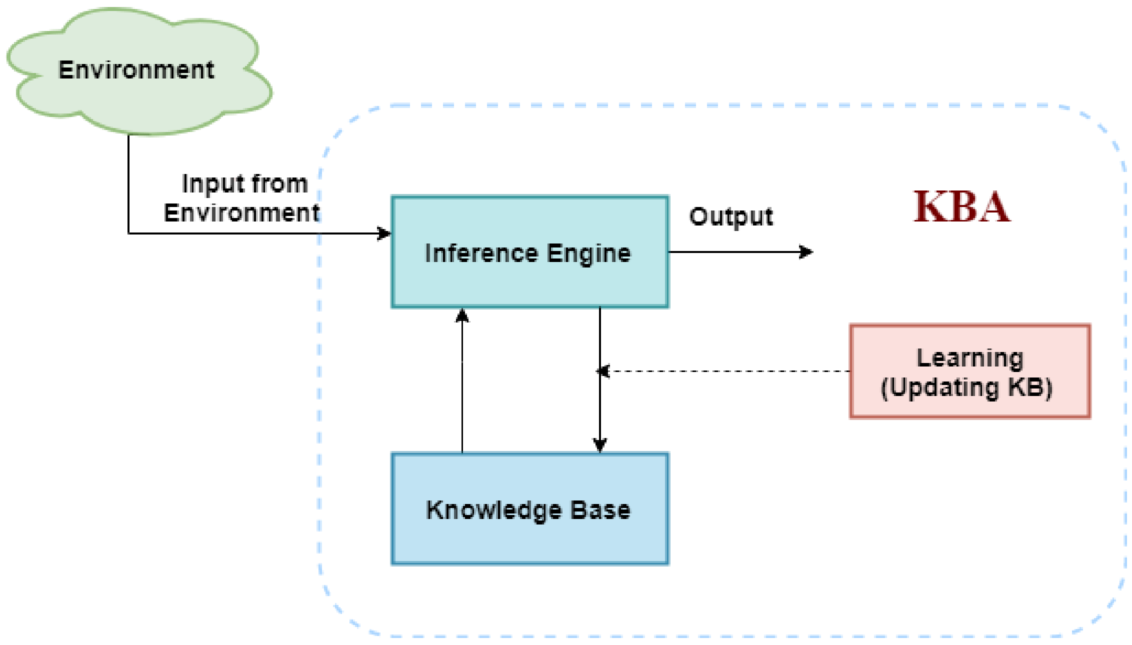
Knowledge-Based Agent in Artificial intelligence

- An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.
- Knowledge-based agents are composed of two main parts:
  - **Knowledge-base and**
  - **Inference system**.

A knowledge-based agent must able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent Should be able to incorporate new percepts
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:



Inference system

- Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.
- Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:

- **Forward chaining**
- **Backward chaining**

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. But how machines do all these things comes under knowledge representation and reasoning. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

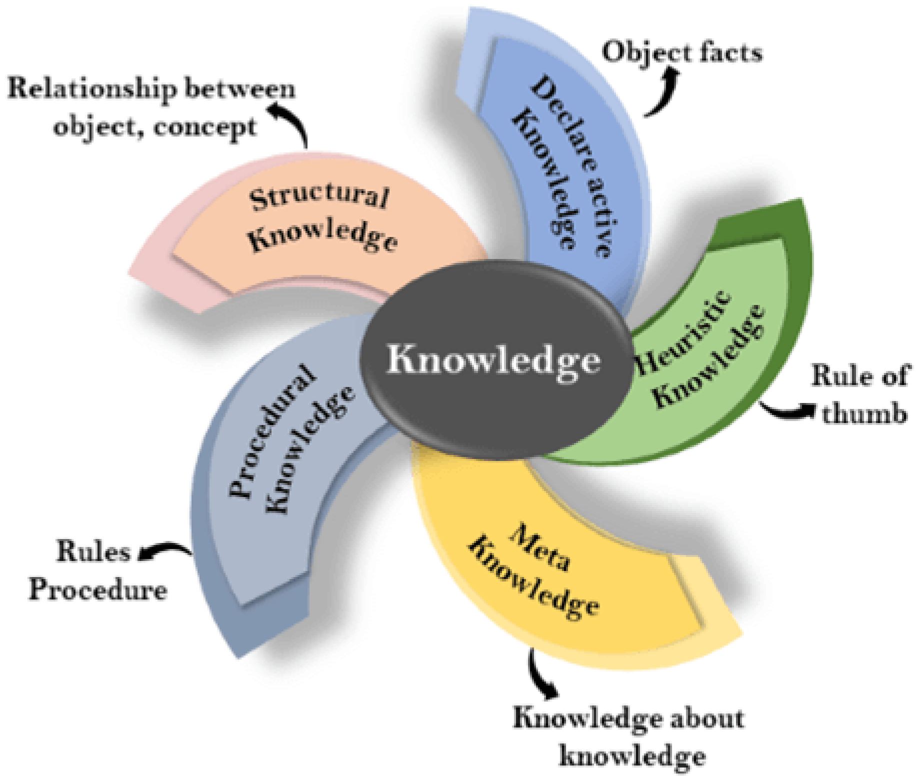
Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

Following are the various types of knowledge:



1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarativesentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

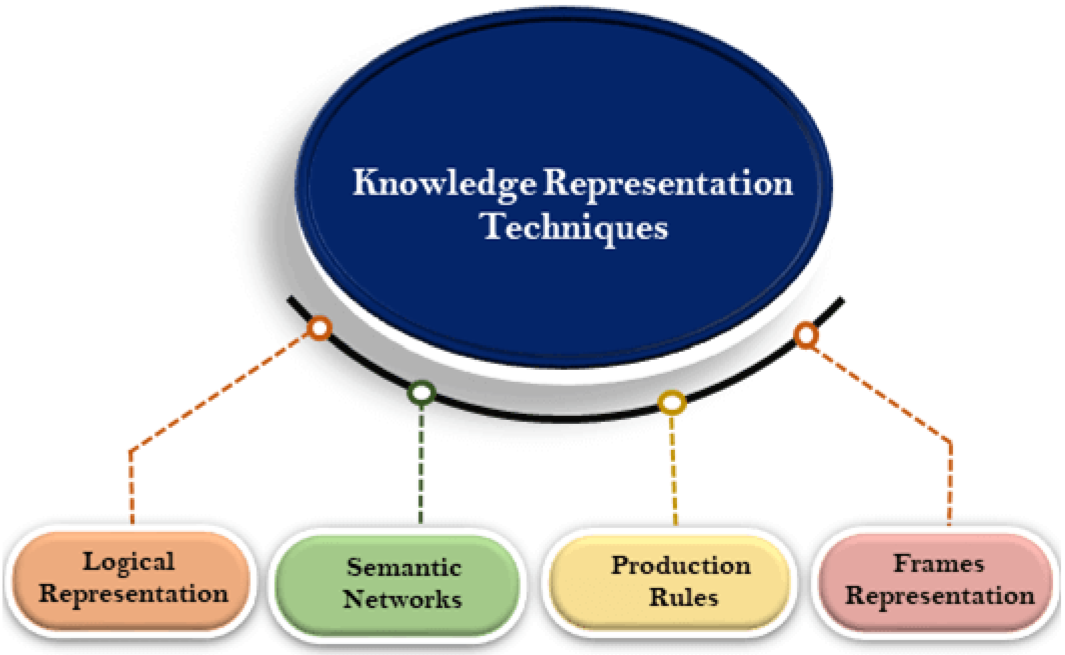
5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



1. Logical Representation

- Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation.
- Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules.

- It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

1. Propositional Logics
2. Predicate logics

Note: We will discuss Prepositional Logics and Predicate logics in later chapters.

Advantages of logical representation:

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

Note: Do not be confused with logical representation and logical reasoning as logical representation is a representation language and reasoning is a process of thinking logically.

2. Semantic Network Representation

- Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks.
- This network consists of nodes representing objects and arcs which describe the relationship between those objects.
- Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

This representation consist of mainly two types of relations:

1. IS-A relation (Inheritance)
2. Kind-of-relation

Advantages of Semantic network:

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

3. Frame Representation

- ➢ A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world.
- ➢ Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations.
- ➢ It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

Example:

- IF (at bus stop AND bus arrives) THEN action (get into the bus)
- IF (on the bus AND paid AND empty seat) THEN action (sit down).
- IF (on bus AND unpaid) THEN action (pay charges).
- IF (bus arrives at destination) THEN action (get down from the bus).

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

Propositional logic in Artificial intelligence

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c) 3+3= 7(False proposition)
4. d) 5 is a prime number.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

1. **Atomic Propositions**
2. **Compound propositions**

- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

1. a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.
2. b) "The Sun is cold" is also a proposition as it is a **false** fact.

- **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

1. a) "It is raining today, and street is wet."
2. b) "Ankit is a doctor, and his clinic is in Mumbai."

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Properties of Operators:

- **Commutativity:**
  - P∧ Q= Q ∧ P, or
  - P ∨ Q = Q ∨ P.
- **Associativity:**
  - (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
  - (P ∨ Q) ∨ R= P ∨ (Q ∨ R)
- **Identity element:**
  - P ∧ True = P,
  - P ∨ True= True.
- **Distributive:**
  - P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
  - P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).
- **DE Morgan's Law:**
  - ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
  - ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).
- **Double-negation elimination:**
  - ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
    - **All the girls are intelligent.**
    - **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Resolution in FOL

Resolution

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.
- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements.
- Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

1. **John likes all kind of food.**
2. **Apple and vegetable are food**
3. **Anything anyone eats and not killed is food.**
4. **Anil eats peanuts and still alive**
5. **Harry eats everything that Anil eats.**
   Prove by resolution that:
6. **John likes peanuts.**

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

a. ∀x: food(x) → likes(John, x)

b. food(Apple) Λ food(vegetables)

c. ∀x ∀y: eats(x, y) Λ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) Λ alive(Anil).

e. ∀x : eats(Anil, x) → eats(Harry, x)

f. ∀x: ¬ killed(x) → alive(x) } added predicates.

g. ∀x: alive(x) →¬ killed(x)

h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

- **Eliminate all implication (→) and rewrite**
    - ∀x ¬ food(x) V likes(John, x)
    - food(Apple) Λ food(vegetables)
    - ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
    - eats (Anil, Peanuts) Λ alive(Anil)
    - ∀x ¬ eats(Anil, x) V eats(Harry, x)
    - ∀x¬ [¬ killed(x)] V alive(x)
    - ∀x ¬ alive(x) V ¬ killed(x)
    - likes(John, Peanuts).
- **Move negation (¬)inwards and rewrite**
    - ∀x ¬ food(x) V likes(John, x)
    - food(Apple) Λ food(vegetables)
    - ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
    - eats (Anil, Peanuts) Λ alive(Anil)
    - ∀x ¬ eats(Anil, x) V eats(Harry, x)
    - ∀x ¬killed(x)] V alive(x)
    - ∀x ¬ alive(x) V ¬ killed(x)
    - likes(John, Peanuts).

- **Rename variables or standardize variables**
    - ∀x ¬ food(x) V likes(John, x)
    - food(Apple) Λ food(vegetables)
    - ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
    - eats (Anil, Peanuts) Λ alive(Anil)
    - ∀w¬ eats(Anil, w) V eats(Harry, w)
    - ∀g ¬killed(g)] V alive(g)
    - ∀k ¬ alive(k) V ¬ killed(k)
    - likes(John, Peanuts).
- **Eliminate existential instantiation quantifier by elimination.**
  In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.
- **Drop Universal quantifiers.**
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.
    - ¬ food(x) V likes(John, x)
    - food(Apple)
    - food(vegetables)
    - ¬ eats(y, z) V killed(y) V food(z)
    - eats (Anil, Peanuts)
    - alive(Anil)
    - ¬ eats(Anil, w) V eats(Harry, w)
    - killed(g) V alive(g)
    - ¬ alive(k) V ¬ killed(k)
    - likes(John, Peanuts).

Note: Statements "food(Apple) Λ food(vegetables)" and "eats (Anil, Peanuts) Λ alive(Anil)" can be written in two separate statements.

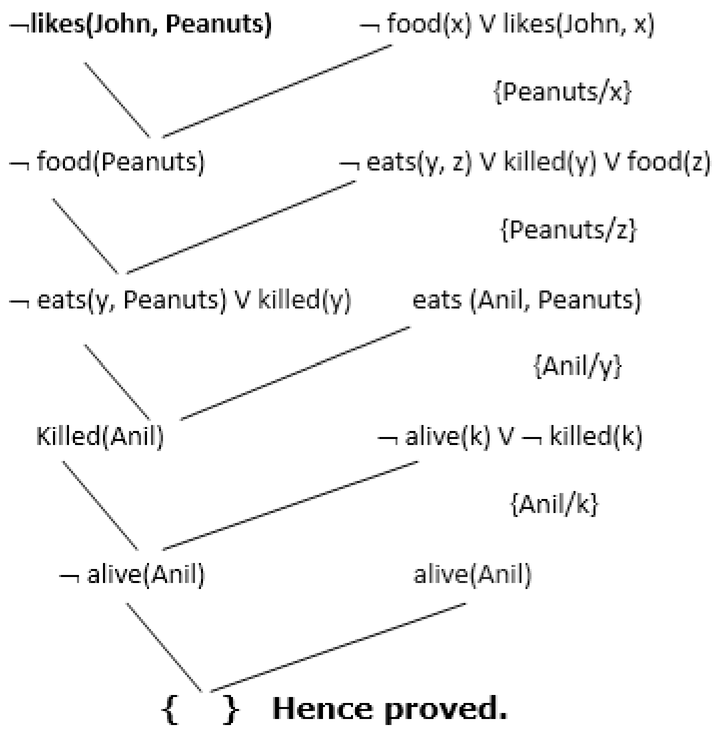
- **Distribute conjunction Λ over disjunction ¬.**
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

1. **Forward chaining**
2. **Backward chaining**

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

B. Backward Chaining:

- Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine.
- A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Bayes' theorem:

- Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.
- In probability theory, it relates the conditional probability and marginal probabilities of two random events.
- Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.
- It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Practical no. – 03

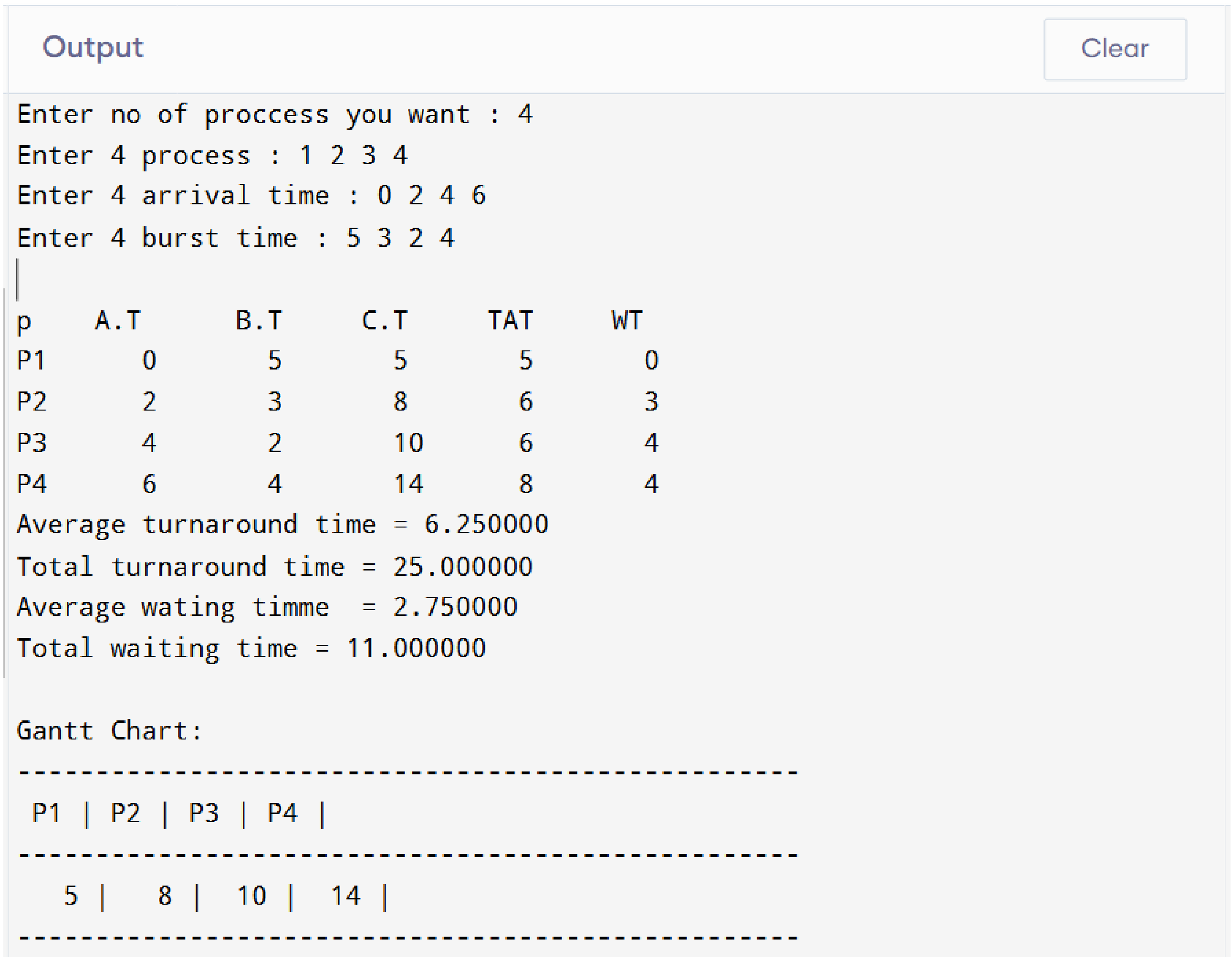
AIM: List of processes / jobs along with their arrival times & CPU burst times is given. Write a program to print the total waiting time, average waiting time, total turnaround time, average turnaround time & Gantt Chart using First Come First Serve (FCFS) CPU scheduling policy.

SOURCE CODE :

```
#include<stdio.h>
int main()
{
   int  p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n;
   float awt=0,atat=0,ttat=0,twt=0;
   printf("Enter no of proccess you want : ");
   scanf("%d",&n);
   printf("Enter %d process : ",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&p[i]);
   }
   printf("Enter %d arrival time : ",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&at[i]);
   }
   printf("Enter %d burst time : ",n);
   for(i=0;i<n;i++)
   {
   scanf("%d",&bt[i]);
   }
   // sorting at,bt, and process according to at
   /* calculating 1st ct */
   ct[0]=at[0]+bt[0];
   /* calculating 2 to n ct */
   for(i=1;i<n;i++)
   {
     //when proess is ideal in between i and i+1
     temp=0;
    if(ct[i-1]<at[i])
    {
      temp=at[i]-ct[i-1];
    }
   ct[i]=ct[i-1]+bt[i]+temp;
   }
   /* calculating tat and wt */
   printf("\np\t A.T\t  B.T\t  C.T\t  TAT\t  WT");
   for(i=0;i<n;i++)
   {
tat[i]=ct[i]-at[i];
wt[i]=tat[i]-bt[i];
atat+=tat[i];
awt+=wt[i];
}
atat=atat/n;
awt=awt/n;
ttat = atat * n;
twt = awt * n;
for(i=0;i<n;i++)
{
printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);
}
printf("\nAverage turnaround time = %f",atat);
printf("\nTotal turnaround time = %f",ttat);
printf("\nAverage wating timme  = %f",awt);
printf("\nTotal waiting time = %f",twt);


  // Gantt Chart
   printf("\n\nGantt Chart:\n");


printf("--------------------------------------------------\n");
```

```
    // Printing the process in the Gantt chart format
    for(i = 0; i < n; i++) {
        printf(" P%d |", p[i]);
    }
    printf("\n-------------------------------------------------\n");
    // Printing the completion times
    for(i = 0; i < n; i++) {
        printf(" %3d |", ct[i]);
    }
    printf("\n-------------------------------------------------\n");
return 0;
}
```

OUTPUT:



```
Enter no of proccess you want : 4
Enter 4 process : 1 2 3 4
Enter 4 arrival time : 0 2 4 6
Enter 4 burst time : 5 3 2 4

p    A.T      B.T     C.T     TAT     WT
P1      0        5       5       5       0
P2      2        3       8       6       3
P3      4        2       10      6       4
P4      6        4       14      8       4
Average turnaround time = 6.250000
Total turnaround time = 25.000000
Average wating timme  = 2.750000
Total waiting time = 11.000000

Gantt Chart:
--------------------------------------------------
 P1 | P2 | P3 | P4 |
--------------------------------------------------
   5 |    8 |   10 |   14 |
--------------------------------------------------
```

Name: Ankit raj Roll No.:-11233036 Department/Semester/Group: CSE/4/G1

Practical no. – 04

AIM: List of processes / jobs along with their arrival times & CPU burst times is given. Write a program to print the total waiting time, average waiting time, total turnaround time, average turnaround time & Gantt Chart using Shortest Job First (SJF) CPU scheduling policy.

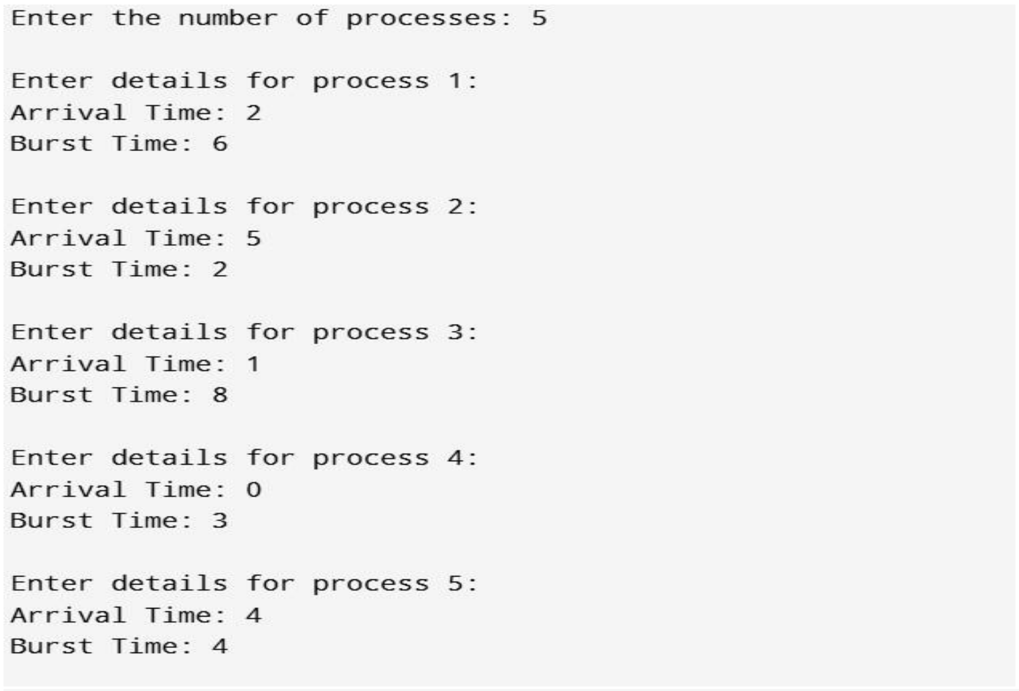
SOURCE CODE:

```
#include <stdio.h>
#define MAX 10
struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int completion_time;
    int  turnaround_time;
    int waiting_time;
    int start_time;
    int response_time;
    int is_completed;
};

// Function to sort processes by arrival time
void sort_by_arrival_time(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].arrival_time > p[j].arrival_time) {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

// Function to calculate times and generate Gantt chart
void calculate_times(struct Process p[], int n) {
    int time = 0;
    int completed = 0;
    int gantt_chart[MAX] = {0};
    int gantt_index = 0;

    float total_waiting_time = 0;
    float total_turnaround_time = 0;
    float total_response_time = 0;

    printf("\nGantt Chart:\n");
    printf("+");
    for (int i = 0; i < n; i++) {
        printf(" ----+");
    }
    printf("\n");

    while (completed < n) {
        int index = -1;
        int min_burst_time = 9999;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival_time <= time && p[i].is_completed == 0) {
                if (p[i].burst_time < min_burst_time) {
                    min_burst_time = p[i].burst_time;
                    index = i;
                }
                if (p[i].burst_time == min_burst_time) {
                    if (p[i].arrival_time < p[index].arrival_time) {
                        index = i;
                    }
                }
            }
        }

        if (index == -1) {
            time++;
        } else {
            p[index].start_time = time;
            p[index].completion_time = time + p[index].burst_time;
            p[index].turnaround_time = p[index].completion_time - p[index].arrival_time;
            p[index].waiting_time = p[index].turnaround_time - p[index].burst_time;
            p[index].response_time = p[index].start_time - p[index].arrival_time;
            p[index].is_completed = 1;

            total_waiting_time += p[index].waiting_time;
            total_turnaround_time += p[index].turnaround_time;
            total_response_time += p[index].response_time;

            gantt_chart[gantt_index++] = time;

            printf("| P%d ", p[index].id);
            time += p[index].burst_time;
            completed++;
        }
    }
```

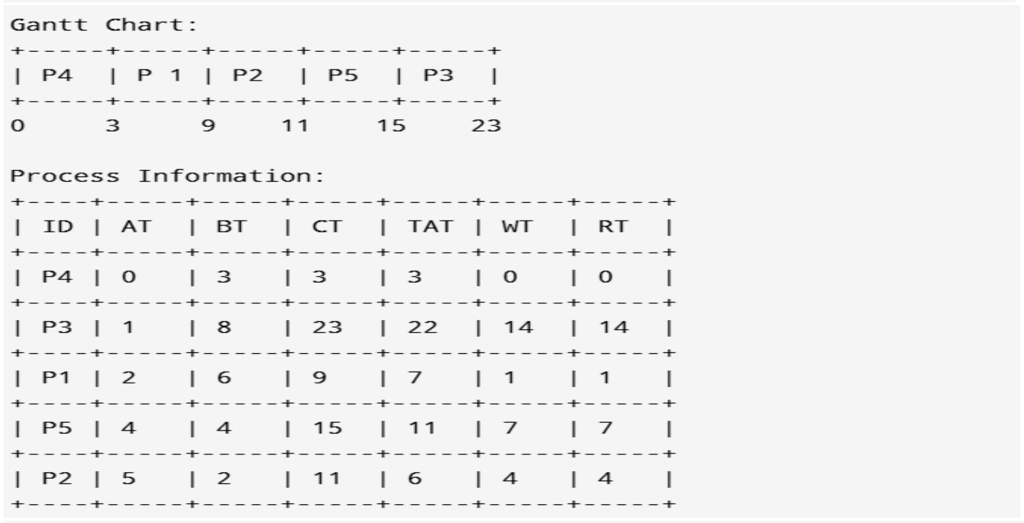
```
gantt_chart[gantt_index] = time;

printf("|\n");
printf("+");
for (int i = 0; i < n; i++) {
    printf(" ----+");
}
printf("\n");

for (int i = 0; i <= gantt_index; i++) {
    printf("%3d  ", gantt_chart[i]);
}
printf("\n");

printf("\nProcess Information:\n");
printf("+---+----+----+----+----+----+----+\n");
printf("| ID | AT  | BT  | CT  | TAT | WT  | RT  |\n");
printf("+---+----+----+----+----+----+----+\n");

for (int i = 0; i < n; i++) {
    printf("| P%-2d| %-3d| %-3d| %-3d| %-3d| %-3d| %-3d|\n",
           p[i].id, p[i].arrival_time, p[i].burst_time,
           p[i].completion_time, p[i].turnaround_time,
           p[i].waiting_time, p[i].response_time);
    printf("+---+----+------+----+----+----+----+\n");
}

printf("\nTotal Waiting Time: %.2f\n", total_waiting_time);
printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
```

```
    printf("Total Turnaround Time: %.2f\n", total_turnaround_time);
    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Total Response Time: %.2f\n", total_response_time);
    printf("Average Response Time: %.2f\n", total_response_time / n);
}

int main() {
    struct Process p[MAX];
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("\nEnter details for process %d:\n", i + 1);
        p[i].id = i + 1;
        printf("Arrival Time: ");
        scanf("%d", &p[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &p[i].burst_time);
        p[i].is_completed = 0;
    }

    sort_by_arrival_time(p, n);
    calculate_times(p, n);

    return 0;
}
```

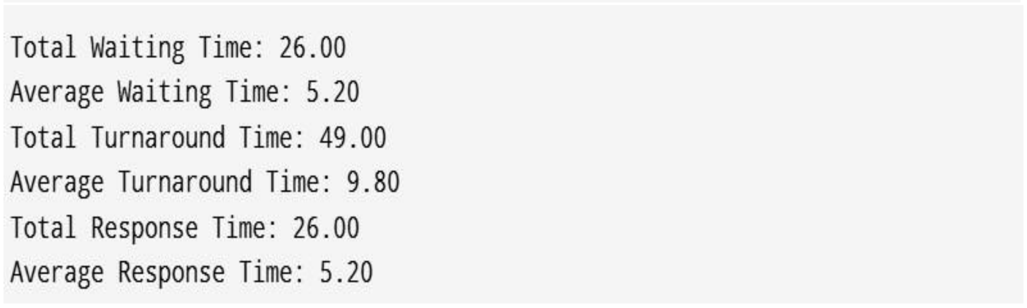
OUTPUT:



```
Enter the number of processes: 5

Enter details for process 1:
Arrival Time: 2
Burst Time: 6

Enter details for process 2:
Arrival Time: 5
Burst Time: 2

Enter details for process 3:
Arrival Time: 1
Burst Time: 8

Enter details for process 4:
Arrival Time: 0
Burst Time: 3

Enter details for process 5:
Arrival Time: 4
Burst Time: 4
```



```
Gantt Chart:
+-----+-----+-----+-----+-----+
| P4  | P 1 | P2  | P5  | P3  |
+-----+-----+-----+-----+-----+
0     3     9     11    15    23

Process Information:
+----+-----+-----+-----+-----+-----+-----+
| ID | AT  | BT  | CT  | TAT | WT  | RT  |
+----+-----+-----+-----+-----+-----+-----+
| P4 | 0   | 3   | 3   | 3   | 0   | 0   |
+----+-----+-----+-----+-----+-----+-----+
| P3 | 1   | 8   | 23  | 22  | 14  | 14  |
+----+-----+-----+-----+-----+-----+-----+
| P1 | 2   | 6   | 9   | 7   | 1   | 1   |
+----+-----+-----+-----+-----+-----+-----+
| P5 | 4   | 4   | 15  | 11  | 7   | 7   |
+----+-----+-----+-----+-----+-----+-----+
| P2 | 5   | 2   | 11  | 6   | 4   | 4   |
+----+-----+-----+-----+-----+-----+-----+
```

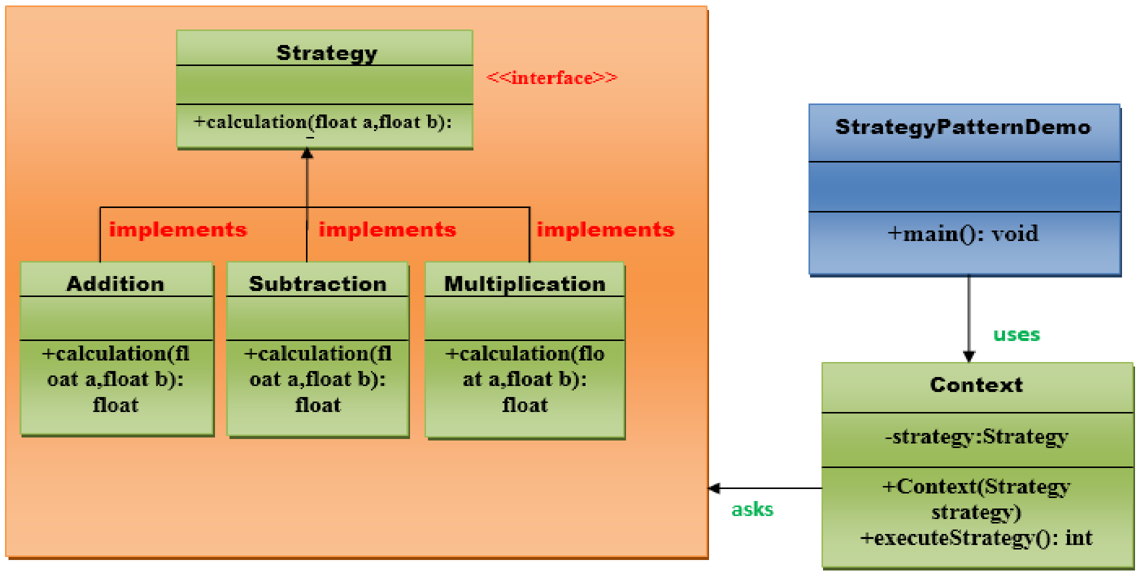


```
Total Waiting Time: 26.00
Average Waiting Time: 5.20
Total Turnaround Time: 49.00
Average Turnaround Time: 9.80
Total Response Time: 26.00
Average Response Time: 5.20
```

Name: Ankit raj Roll No.:-11233036 Department/Semester/Group: CSE/4/G1

EXPERIMENT NO :- 03

Aim: **Write program to understand The Strategy Pattern**.



Implementation of Strategy Pattern:

Step 1:

Create a Strategy interface.

.

```
public interface Strategy {
   public float calculation(float a, float b);
```

.

Step 2:

Create a Addition class that will implement Startegy interface.

```
.
public class Addition implements Strategy{

   @Override
   public float calculation(float a, float b) {
      return a+b;
   }

.
```

Step 3:

Create a Subtraction class that will implement Startegy interface.

```
//This is a class.
public class Subtraction  implements Strategy{

   @Override
   public float calculation(float a, float b) {
      return a-b;
   }
```

Step 4:

Create a Multiplication class that will implement Startegy interface.

```
//This is a class.

public class Multiplication implements Strategy{

   @Override
   public float calculation(float a, float b){
   }
.
```

Step 5:

Create a Context class that will ask from Startegy interface to execute the type of strategy.

```
.  public class Context {

      private Strategy strategy;
```

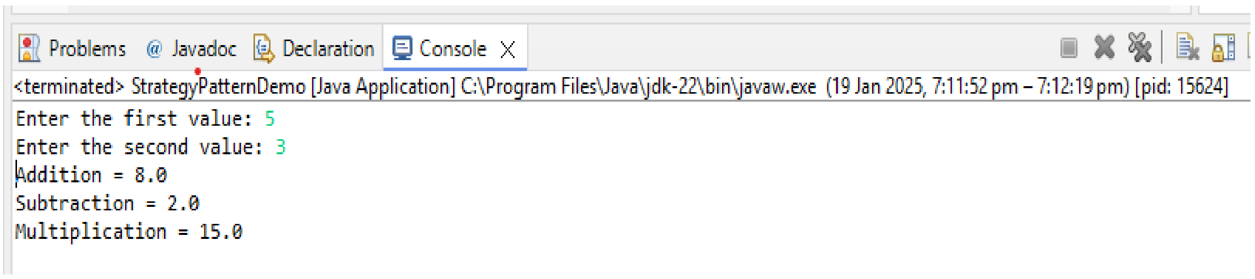
```
    public Context(Strategy strategy){
       this.strategy = strategy;
    }

    public float executeStrategy(float num1, float num2){
       return strategy.calculation(num1, num2);
    }
}
```

Step 6:

Create a StartegyPatternDemo class.

```
.
import java.io.BufferedReader;
import java.io.IOException;  import java.io.InputStreamReader;

public class StrategyPatternDemo {
   public static void main(String[] args) throws NumberFormatException, IOExcept
ion {

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in)
);
        System.out.print("Enter the first value: ");
        float value1=Float.parseFloat(br.readLine());
        System.out.print("Enter the second value: ");
        float value2=Float.parseFloat(br.readLine());
        Context context = new Context(new Addition());
        System.out.println("Addition = " + context.executeStrategy(value1, value2));
        context = new Context(new Subtraction());
        System.out.println("Subtraction = " + context.executeStrategy(value1, value2))
;

        context = new Context(new Multiplication());
            System.out.println("Multiplication = " + context.executeStrategy(value1, val
    ue2))  .
```

OUTPUT:-



```
<terminated> StrategyPatternDemo [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (19 Jan 2025, 7:11:52 pm – 7:12:19 pm) [pid: 15624]
Enter the first value: 5
Enter the second value: 3
Addition = 8.0
Subtraction = 2.0
Multiplication = 15.0
```