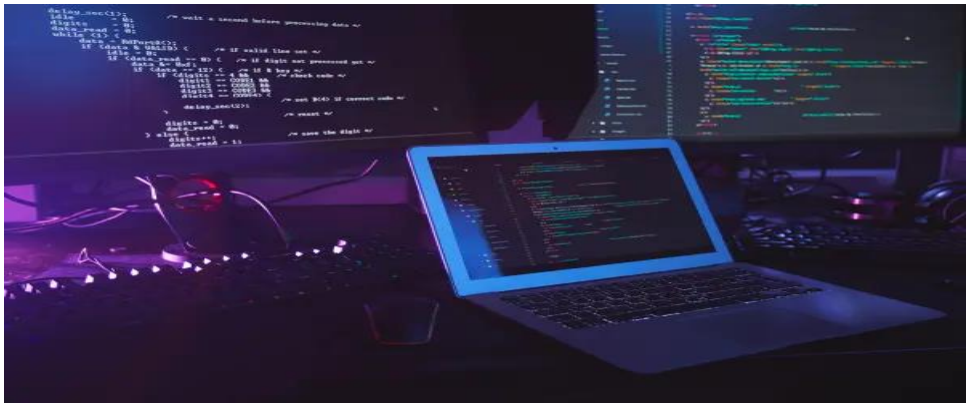
NUST
NATIONAL UNIVERSITY
OF SCIENCES & TECHNOLOGY

Fop project II

Submitted to: DR. SAQIB NAZIR

GROUP MEMBERS

SR NO	CMS	NAME
01	465779	OMAR MUDASSAR
02	477228	MUHAMMAD UZAIR
03	471240	HASHIR TARIQ
04		FAIZAN ALI
05	463926	FAIEZ AHMAD



SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING

Importing Necessary Libraries

```
python
Copy code
import feedparser
from datetime import datetime
```

- **`feedparser`**: This module is used to parse RSS feeds. It can read and parse feeds from various websites, allowing you to extract the information in a structured format.
- **`datetime`**: This module provides classes for manipulating dates and times. It will be used to handle publication dates of news stories.

Class to Represent a News Story

```
python
Copy code
class NewsStory:
  def __init__(self, guid, title, description, link, pubdate):
    self.guid = guid
    self.title = title
    self.description = description
    self.link = link
    self.pubdate = pubdate

  def get_title(self):
    return self.title

  # Similar methods for other attributes (description, link, etc.)
```

- **`NewsStory`**: This class represents a news story with its attributes.
  - **`__init__` method**: The constructor initializes the `NewsStory` object with the following parameters:
    - `guid`: A unique identifier for the story.
    - `title`: The title of the news story.
    - `description`: A brief description or summary of the story.
    - `link`: A URL linking to the full news story.
    - `pubdate`: The publication date of the story.
  - **`get_title` method**: This method returns the title of the news story. Similar methods for other attributes (`description`, `link`, `pubdate`) should be implemented to access those attributes.

Base Class for Triggers (Abstract)

```
python
Copy code
class Trigger:
  def evaluate(self, story):
    raise NotImplementedError("Subclass must implement evaluate")
```

- **`Trigger`**: This is an abstract base class for all triggers.

    - **`evaluate` method**: This is an abstract method that must be implemented by any subclass. It raises a `NotImplementedError` if called directly. The method is intended to evaluate a `NewsStory` object and determine whether the trigger conditions are met.

Trigger for Matching a Phrase in the Title (Case-Insensitive)

```
python
Copy code
class TitleTrigger(Trigger):
  def __init__(self, phrase):
    self.phrase = phrase.lower()

  def evaluate(self, story):
    return self.phrase in story.get_title().lower()
```

- **`TitleTrigger`**: A subclass of `Trigger` that fires when a specified phrase is found in the title of a news story.
    - **`__init__` method**: The constructor initializes the trigger with a `phrase`, converting it to lowercase to ensure case-insensitive matching.
    - **`evaluate` method**: This method checks if the lowercase phrase is present in the lowercase title of the `NewsStory`. If the phrase is found, the method returns `True`; otherwise, it returns `False`.

Similar Classes for Other Triggers

You would implement similar classes for other triggers such as `DescriptionTrigger`, `BeforeTrigger`, `AfterTrigger`, etc., following the same pattern as `TitleTrigger`.

Trigger that Inverts the Output of Another Trigger

```
python
Copy code
class NotTrigger(Trigger):
  def __init__(self, trigger):
    self.trigger = trigger

  def evaluate(self, story):
    return not self.trigger.evaluate(story)
```

- **`NotTrigger`**: A trigger that inverts the result of another trigger.
    - **`__init__` method**: The constructor initializes the trigger with another `trigger` object.
    - **`evaluate` method**: This method returns the opposite of the result of the enclosed trigger's `evaluate` method. If the enclosed trigger returns `True`, this method returns `False`, and vice versa.

Trigger that Fires Only if Both Sub-Triggers Fire (AND)

```
python
Copy code
class AndTrigger(Trigger):
  def __init__(self, trigger1, trigger2):
    self.trigger1 = trigger1
    self.trigger2 = trigger2

  def evaluate(self, story):
    return self.trigger1.evaluate(story) and self.trigger2.evaluate(story)
```

- **`AndTrigger`**: A trigger that fires only if both sub-triggers fire.
  - **`__init__` method**: The constructor initializes the trigger with two sub-triggers (`trigger1` and `trigger2`).
  - **`evaluate` method**: This method returns `True` only if both sub-triggers' `evaluate` methods return `True`.

Trigger that Fires if At Least One Sub-Trigger Fires (OR)

```
python
Copy code
class OrTrigger(Trigger):
  def __init__(self, trigger1, trigger2):
    self.trigger1 = trigger1
    self.trigger2 = trigger2

  def evaluate(self, story):
    return self.trigger1.evaluate(story) or self.trigger2.evaluate(story)
```

- **`OrTrigger`**: A trigger that fires if at least one of the sub-triggers fires.
  - **`__init__` method**: The constructor initializes the trigger with two sub-triggers (`trigger1` and `trigger2`).
  - **`evaluate` method**: This method returns `True` if either of the sub-triggers' `evaluate` methods return `True`.

Function to Parse Time String from Configuration File

```
python
Copy code
def parse_time(time_str):
  return datetime.strptime(time_str, "%d %b %Y %H:%M:%S")
```

- **`parse_time`**: This function converts a time string in the format "3 Oct 2023 17:00:10" to a `datetime` object using `datetime.strptime`.

Function to Parse Trigger Definitions from Configuration File

```
python
Copy code
def read_trigger_config(filename):
  trigger_map = {
```

```
    "TITLE": TitleTrigger,
    "DESCRIPTION": DescriptionTrigger,
    "BEFORE": BeforeTrigger,
    "AFTER": AfterTrigger,
    "NOT": NotTrigger,
    "AND": AndTrigger,
    "OR": OrTrigger,
  }
  triggers = {}
  trigger_list = []

  with open(filename, 'r') as f:
    for line in f:
      line = line.strip()
      if not line or line.startswith('//'):
        continue

      parts = line.split(',')
      trigger_name = parts[0]

      if trigger_name == "ADD":
        for name in parts[1:]:
          trigger_list.append(triggers[name])
      else:
        trigger_type = parts[1]
        if trigger_type in {"AND", "OR"}:
          triggers[trigger_name] =
trigger_map[trigger_type](triggers[parts[2]], triggers[parts[3]])
        elif trigger_type in {"NOT"}:
          triggers[trigger_name] =
trigger_map[trigger_type](triggers[parts[2]])
        elif trigger_type in {"BEFORE", "AFTER"}:
          date = parse_time(parts[2])
          triggers[trigger_name] = trigger_map[trigger_type](date)
        else:
          triggers[trigger_name] = trigger_map[trigger_type](parts[2])

  return trigger_list
```

- **`read_trigger_config`**: This function reads trigger definitions from a configuration file and creates trigger objects.
    - **`trigger_map`**: A dictionary mapping trigger type strings to their corresponding classes.
    - **`triggers`**: A dictionary to store created triggers by their names.
    - **`trigger_list`**: A list to store the final set of triggers to be used.
    - The file is read line by line:
        - Lines starting with `//` are comments and ignored.
        - Lines starting with `ADD` add the specified triggers to the final list.
        - Other lines define triggers based on their type and parameters, creating the corresponding trigger objects.

Function to Filter Stories Based on Trigger List

```
python
Copy code
def filter_stories(stories, triggers):
  filtered_stories = []
  for story in stories:
    for trigger in triggers:
      if trigger.evaluate(story):
        filtered_stories.append(story)
        break  # Stop checking triggers if any match
  return filtered_stories
```

- **`filter_stories`**: This function filters a list of news stories based on a list of triggers.
  - Iterates through each story and each trigger.
  - Adds the story to the `filtered_stories` list if any trigger fires for that story, then breaks to avoid checking further triggers for that story.

Function to Fetch News Stories from RSS Feeds (Implementation Omitted)

```
python
Copy code
def get_stories():
  # Placeholder for actual implementation
  pass
```

- **`get_stories`**: Placeholder function to fetch news stories from RSS feeds. The actual implementation would use `feedparser` to parse RSS feeds and create `NewsStory` objects.

Main Function

```
python
Copy code
def main():
  # Get triggers from configuration file
  triggers = read_trigger_config("triggers.txt")

  # Fetch news stories
  stories = get_stories()

  # Filter stories based on triggers
  filtered_stories = filter_stories(stories, triggers)

  # Display filtered stories to the user (implementation omitted)
  for story in filtered_stories:
    print(story.get_title())

if __name__ == "__main__":
  main()
```

- **`main`**: The main function to run the script.
  - Reads triggers from the configuration file using `read_trigger_config`.
  - Fetches news stories using `get_stories`.

- Filters the stories based on the triggers using `filter_stories`.
- Displays the filtered stories to the user by printing their titles. (You would implement a more sophisticated display method in a real application.)
- The script runs the `main` function if executed directly (i.e., not imported as a module) by checking `if __name__ == "__main__":`

PART-2:

Importing Necessary Libraries

```
python
Copy code
from datetime import datetime
```

- **`datetime`**: This module provides classes for manipulating dates and times. It's used here to parse and compare publication dates of news stories.

Class to Represent a News Story

```
python
Copy code
class NewsStory:
  def __init__(self, guid, title, description, link, pubdate):
    self.guid = guid
    self.title = title
    self.description = description
    self.link = link
    self.pubdate = pubdate

  def get_title(self):
    return self.title

  # Similar methods for other attributes (description, link, etc.)
```

- **`NewsStory`**: This class represents a news story with several attributes.
  - **`__init__` method**: The constructor initializes a `NewsStory` object with:
    - `guid`: A unique identifier for the story.
    - `title`: The title of the news story.
    - `description`: A brief description of the story.
    - `link`: A URL linking to the full news story.
    - `pubdate`: The publication date of the story.
  - **`get_title` method**: This method returns the title of the news story.
  - **Placeholder comment**: Indicates that similar methods should be implemented to access other attributes (`description`, `link`, `pubdate`).

Base Class for Triggers (Abstract)

```
python
Copy code
```

```
class Trigger:
  def evaluate(self, story):
    raise NotImplementedError("Subclass must implement evaluate")
```

- **`Trigger`**: This is an abstract base class for all triggers.
  - **`evaluate` method**: An abstract method that raises a `NotImplementedError`. Subclasses must implement this method to define the trigger's behavior.

Trigger for Matching a Phrase in the Title (Case-Insensitive)

```
python
Copy code
class TitleTrigger(Trigger):
  def __init__(self, phrase):
    self.phrase = phrase.lower()

  def evaluate(self, story):
    return self.phrase in story.get_title().lower()
```

- **`TitleTrigger`**: A subclass of `Trigger` that fires when a specified phrase is found in the title of a news story.
  - **`__init__` method**: The constructor initializes the trigger with a `phrase`, converting it to lowercase for case-insensitive matching.
  - **`evaluate` method**: This method checks if the lowercase phrase is present in the lowercase title of the `NewsStory`. Returns `True` if the phrase is found; otherwise, returns `False`.

Trigger for Matching a Phrase in the Description (Case-Insensitive)

```
python
Copy code
class DescriptionTrigger(Trigger):
  def __init__(self, phrase):
    self.phrase = phrase.lower()

  def evaluate(self, story):
    return self.phrase in story.get_description().lower()
```

- **`DescriptionTrigger`**: Similar to `TitleTrigger`, but it checks the description instead of the title.
  - **`__init__` method**: Initializes the trigger with a `phrase`, converting it to lowercase.
  - **`evaluate` method**: Checks if the lowercase phrase is present in the lowercase description of the `NewsStory`.

Trigger to Check if Publish Date is Before a Specific Time

```
python
Copy code
class BeforeTrigger(Trigger):
```

```
  def __init__(self, before_time):
    self.before_time = before_time

  def evaluate(self, story):
    story_date = datetime.strptime(story.pubdate, "%a, %d %b %Y %H:%M:%S %Z")
    return story_date < self.before_time
```

- **`BeforeTrigger`**: A trigger that fires if the publication date of the story is before a specified time.
    - **`__init__` method**: Initializes the trigger with a `before_time`, which is a `datetime` object.
    - **`evaluate` method**: Parses the publication date of the story into a `datetime` object and checks if it is before the `before_time`.

Trigger to Check if Publish Date is After a Specific Time

```
python
Copy code
class AfterTrigger(Trigger):
  def __init__(self, after_time):
    self.after_time = after_time

  def evaluate(self, story):
    story_date = datetime.strptime(story.pubdate, "%a, %d %b %Y %H:%M:%S %Z")
    return story_date > self.after_time
```

- **`AfterTrigger`**: Similar to `BeforeTrigger`, but it fires if the publication date is after a specified time.
    - **`__init__` method**: Initializes the trigger with an `after_time`, which is a `datetime` object.
    - **`evaluate` method**: Parses the publication date of the story and checks if it is after the `after_time`.

Trigger that Inverts the Output of Another Trigger

```
python
Copy code
class NotTrigger(Trigger):
  def __init__(self, trigger):
    self.trigger = trigger

  def evaluate(self, story):
    return not self.trigger.evaluate(story)
```

- **`NotTrigger`**: A trigger that inverts the result of another trigger.
    - **`__init__` method**: Initializes the trigger with another `trigger` object.
    - **`evaluate` method**: Returns the opposite of the result of the enclosed trigger's `evaluate` method.

Trigger that Fires Only if Both Sub-Triggers Fire (AND)

```
python
Copy code
class AndTrigger(Trigger):
  def __init__(self, trigger1, trigger2):
    self.trigger1 = trigger1
    self.trigger2 = trigger2

  def evaluate(self, story):
    return self.trigger1.evaluate(story) and self.trigger2.evaluate(story)
```

- **`AndTrigger`**: A trigger that fires if both sub-triggers fire.
  - **`__init__` method**: Initializes the trigger with two sub-triggers (`trigger1` and `trigger2`).
  - **`evaluate` method**: Returns `True` if both sub-triggers' `evaluate` methods return `True`.

Trigger that Fires if At Least One Sub-Trigger Fires (OR)

```
python
Copy code
class OrTrigger(Trigger):
  def __init__(self, trigger1, trigger2):
    self.trigger1 = trigger1
    self.trigger2 = trigger2

  def evaluate(self, story):
    return self.trigger1.evaluate(story) or self.trigger2.evaluate(story)
```

- **`OrTrigger`**: A trigger that fires if at least one of the sub-triggers fires.
  - **`__init__` method**: Initializes the trigger with two sub-triggers (`trigger1` and `trigger2`).
  - **`evaluate` method**: Returns `True` if either of the sub-triggers' `evaluate` methods return `True`.

Function to Parse Time String from Configuration File

```
python
Copy code
def parse_time(time_str):
  # Adjust this based on your specific time format in the config file
  return datetime.strptime(time_str, "%Y-%m-%d %H:%M:%S")
```

- **`parse_time`**: This function converts a time string in the format "YYYY-MM-DD HH:MM:SS" to a `datetime` object using `datetime.strptime`.

Function to Parse Trigger Definitions from Configuration File

```
python
Copy code
def read_trigger_config(filename):
```

```
  triggers = []
  with open(filename, 'r') as f:
    for line in f:
      # Strip comments starting with '#'
      line = line.strip().split('#')[0].strip()
      if not line:  # Skip empty lines
        continue
      parts = line.split(':')
      trigger_type, value = parts[0].lower(), parts[1].strip()
      if trigger_type == "title":
        triggers.append(TitleTrigger(value))
      elif trigger_type == "description":
        triggers.append(DescriptionTrigger(value))
      elif trigger_type == "before":
        triggers.append(BeforeTrigger(parse_time(value)))
      elif trigger_type == "after":
        triggers.append(AfterTrigger(parse_time(value)))
      elif trigger_type == "not":
        # Ensure the trigger to be inverted exists first
        if triggers:  # Check if there are triggers available before
inverting
          triggers.append(NotTrigger(triggers[-1]))
  return triggers
```

- **`read_trigger_config`**: This function reads trigger definitions from a configuration file and creates trigger objects.
  - **`triggers`**: An empty list to store created triggers.
  - **File reading**: Opens the file specified by `filename` and reads it line by line.
  - **Comment stripping**: Removes comments (anything after `#`) and strips whitespace.
  - **Skipping empty lines**: Skips lines that are empty after stripping.
  - **Parsing lines**: Splits each line into parts based on `:`. The first part is the trigger type, and the second part is the value.
  - **Creating triggers**: Based on the trigger type, it creates the appropriate trigger and adds it to the `triggers` list.
  - **`NotTrigger` handling**: Ensures there are triggers available to invert before adding a `NotTrigger`.
  - **Returning triggers**: Returns the list of created triggers.

PART-3:

Importing Necessary Libraries

```
python
Copy code
from abc import ABC, abstractmethod
from datetime import datetime
```

- **`abc`**: Provides tools for defining Abstract Base Classes (ABCs) in Python.

- **`datetime`**: Provides classes for manipulating dates and times.

Class to Represent a News Story

```
python
Copy code
class NewsStory:
    def __init__(self, guid, title, description, link, pubdate):
        self.guid = guid
        self.title = title
        self.description = description
        self.link = link
        self.pubdate = pubdate

    def get_guid(self):
        return self.guid

    def get_title(self):
        return self.title

    def get_description(self):
        return self.description

    def get_link(self):
        return self.link

    def get_pubdate(self):
        return self.pubdate
```

- **`NewsStory`**: This class represents a news story with several attributes.
    - **`__init__` method**: Initializes a `NewsStory` object with `guid`, `title`, `description`, `link`, and `pubdate`.
    - **Getter methods**: These methods return the values of the corresponding attributes (`guid`, `title`, `description`, `link`, `pubdate`).

PhraseTrigger Abstract Class

```
python
Copy code
class PhraseTrigger(ABC):
    def __init__(self, phrase):
        self.phrase = phrase.lower()

    @abstractmethod
    def is_phrase_in(self, text):
        pass
```

- **`PhraseTrigger`**: An abstract base class for triggers that look for phrases in text.
    - **`__init__` method**: Initializes the trigger with a `phrase`, converting it to lowercase.
    - **`is_phrase_in` method**: An abstract method that subclasses must implement to check if the phrase is in the given text.

TitleTrigger Class

```
python
Copy code
class TitleTrigger(PhraseTrigger):
    def is_phrase_in(self, text):
        return self.phrase in text.lower()
```

- **`TitleTrigger`**: A subclass of `PhraseTrigger` that checks if a phrase is in the title of a news story.
    - **`is_phrase_in` method**: Checks if the lowercase phrase is in the lowercase text.

DescriptionTrigger Class

```
python
Copy code
class DescriptionTrigger(PhraseTrigger):
    def is_phrase_in(self, text):
        return self.phrase in text.lower()
```

- **`DescriptionTrigger`**: Similar to `TitleTrigger`, but it checks the description instead of the title.
    - **`is_phrase_in` method**: Checks if the lowercase phrase is in the lowercase text.

TimeTrigger Abstract Class

```
python
Copy code
class TimeTrigger(ABC):
    def __init__(self, time):
        self.time = datetime.strptime(time, "%d %b %Y %H:%M:%S")
```

- **`TimeTrigger`**: An abstract base class for triggers that involve a specific time.
    - **`__init__` method**: Initializes the trigger with a `time`, converting it from a string to a `datetime` object.

BeforeTrigger and AfterTrigger Classes

```
python
Copy code
class BeforeTrigger(TimeTrigger):
    def evaluate(self, news_story):
        return news_story.get_pubdate() < self.time

class AfterTrigger(TimeTrigger):
    def evaluate(self, news_story):
        return news_story.get_pubdate() > self.time
```

- **`BeforeTrigger`**: Fires if the publication date of the story is before the specified time.

    - **`evaluate` method**: Returns `True` if the story's publication date is before the specified time.
- **`AfterTrigger`**: Fires if the publication date is after the specified time.
    - **`evaluate` method**: Returns `True` if the story's publication date is after the specified time.

NOTTrigger Class

```
python
Copy code
class NotTrigger(Trigger):
    def __init__(self, other_trigger):
        self.other_trigger = other_trigger

    def evaluate(self, news_story):
        return not self.other_trigger.evaluate(news_story)
```

- **`NotTrigger`**: Inverts the result of another trigger.
    - **`__init__` method**: Initializes the trigger with another `trigger` object.
    - **`evaluate` method**: Returns the opposite of the enclosed trigger's `evaluate` method.

ANDTrigger Class

```
python
Copy code
class AndTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

    def evaluate(self, news_story):
        return self.trigger1.evaluate(news_story) and
self.trigger2.evaluate(news_story)
```

- **`AndTrigger`**: Fires if both sub-triggers fire.
    - **`__init__` method**: Initializes the trigger with two sub-triggers (`trigger1` and `trigger2`).
    - **`evaluate` method**: Returns `True` if both sub-triggers' `evaluate` methods return `True`.

ORTrigger Class

```
python
Copy code
class OrTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2
```

```
    def evaluate(self, news_story):
        return self.trigger1.evaluate(news_story) or
self.trigger2.evaluate(news_story)
```

- **`OrTrigger`**: Fires if at least one of the sub-triggers fires.
  - **`__init__` method**: Initializes the trigger with two sub-triggers (`trigger1` and `trigger2`).
  - **`evaluate` method**: Returns `True` if either of the sub-triggers' `evaluate` methods return `True`.

Filtering Function

```
python
Copy code
def filter_stories(stories, triggerlist):
    filtered_stories = []
    for story in stories:
        for trigger in triggerlist:
            if trigger.evaluate(story):
                filtered_stories.append(story)
                break
    return filtered_stories
```

- **`filter_stories`**: Filters a list of news stories based on a list of triggers.
  - **`filtered_stories`**: An empty list to store stories that match any trigger.
  - **Loop through stories**: For each story, loop through the triggers.
  - **Evaluate triggers**: If any trigger fires (`evaluate` returns `True`), add the story to `filtered_stories` and stop checking further triggers for that story.
  - **Return filtered stories**: Returns the list of stories that matched any trigger.

Read Trigger Configuration

```
python
Copy code
def read_trigger_config(filename):
    triggers = {}
    triggerlist = []
    with open(filename, 'r') as file:
        for line in file:
            line = line.strip()
            if line.startswith('//') or not line:
                continue
            parts = line.split(',')
            trigger_name = parts[0]
            trigger_type = parts[1]
            if trigger_type == 'TITLE':
                phrase = parts[2]
                triggers[trigger_name] = TitleTrigger(phrase)
            elif trigger_type == 'DESCRIPTION':
                phrase = parts[2]
                triggers[trigger_name] = DescriptionTrigger(phrase)
            elif trigger_type == 'BEFORE':
```

```
                time = parts[2]
                triggers[trigger_name] = BeforeTrigger(time)
            elif trigger_type == 'AFTER':
                time = parts[2]
                triggers[trigger_name] = AfterTrigger(time)
            elif trigger_type == 'NOT':
                other_trigger = parts[2]
                triggers[trigger_name] = NotTrigger(triggers[other_trigger])
            elif trigger_type == 'AND':
                trigger1 = parts [2]
                trigger2 = parts [3]
                triggers[trigger_name] = AndTrigger(triggers[trigger1],
triggers[trigger2])
            elif trigger_type == 'OR':
                trigger1 = parts[2]
                trigger2 = parts[3]
                triggers[trigger_name] = OrTrigger(triggers[trigger1],
triggers[trigger2])
            elif trigger_type == 'ADD':
                for name in parts[1:]:
                    triggerlist.append(triggers[name])
    return triggerlist
```

- **`read_trigger_config`**: Reads a configuration file and creates trigger objects based on its content.
    - **`triggers`**: A dictionary to store created trigger objects.
    - **`triggerlist`**: A list to store triggers to be returned.
    - **File reading**: Opens the file specified by `filename` and reads it line by line.
    - **Strip and skip comments**: Removes comments (anything after `//`) and strips whitespace.
    - **Skipping empty lines**: Skips lines that are empty after stripping.
    - **Parsing lines**: Splits each line into parts based on `,`. The first part is the trigger name, the second part is the trigger type, and the remaining parts are the parameters.
    - **Creating triggers**: Based on the trigger type, it creates the appropriate trigger and adds it to the `triggers` dictionary.
    - **`ADD` handling**: Adds the specified triggers to the `triggerlist`.
    - **Returning triggers**: Returns the list of created triggers.

Test Code

```
python
Copy code
if __name__ == "__main__":
    # Example usage of read_trigger_config
    triggerlist = read_trigger_config('triggers.txt')
    print(triggerlist)  # You can use this trigger list to filter stories
```

- **Test Code**: This section runs only if the script is executed