FORTRAN Style Pseudocode for the CALL FUNCTION

THE CALL FUNCTION

Initialize the compiler

```
clear all ← Clear the previously created variables
close all ← Close all the open windows
warning off; ← Turn off the oversize image display warnings
```

Generate an array of image paths for both objects and scenes

```
scene{1} = 'D:\Research\Image Database\Database 3\slantscene.jpg';
scene{2} = 'D:\Research\Image Database\Database 3\bigscene2.jpg';
scene{3} = 'D:\Research\Image Database\Database 3\bigscene3.jpg';
.
.
And so on

obj{1} = 'D:\Research\Image Database\Database 3\ic.jpg';
obj{2} = 'D:\Research\Image Database\Bunch of Wires.jpg';
obj{3} = 'D:\Research\Image Database\USB Multiplexer.jpg';
.
.
And so on

count_objects = 50; ← Determining the total number of object images in
database.
count_scenes = 8; ← Determining the total number of scene images in database
received for computation externally.
```

Run the algorithm for all the pairs of object and scene images using the **WHILE LOOP**

```
j = 1; ← Counter for the following while loop

while (count) ← For computing one pair of images at a time
```

> **Get** the object image filename and scene image filename by using the "fileparts" algorithm.
>
> **Print** on display the name of object and scene pair being computed.
>
> **CALL** THE FUNCTION keyword_finder with inputs of the paths of object and scenes that returns the matching rate.
>
> **FUNCTION** matching_rate = keyword_finder(object_path,scene_path)
>
> Display on the screen the matching rate of the object and the scene to help determine the presence of object in the scene.
>
> j = j + 1; ← Increment the counter

```
if the matching rate is more than the required threshold then

      Generate a sentence and store that keyword into that

    end

end
```

Sending it further to language processing tools.

After computing the entire set of database, display all keywords of objects present in the scene with both the object name and scene name and also write them to a ".dat" file which can be further processed by the NLTK (Natural Language Toolkit) algorithm.

disp('End of Code'); 🡐 Indicate the termination of program to the user.

FORTRAN Style Pseudocode for the **KEYWORD_FINDER_FUNCTION**

THE KEYWORD_FINDER FUNCTION

DESCRIPTION: The **KEYWORD_FINDER** function is a function that takes two images as input and outputs their matching rate thus enabling the user to determine the presence of any object in any scene.

NOTE: KEYWORD_FINDER function is an **OVERLOADED** function which assumes the name of the input file as the keyword of the image file unless specifically a separate input is provided for it. Similarly by default the developer's mode is disabled unless activated specifically by input.

Inputs received when function is called

1. Object Image Path
2. Scene Image Path
3. Developer's Mode Activation 🡐 Only used for analysis of errors in algorithm

DEVELOPER's MODE: Because of the complexity of the algorithm and variation of behavior of the algorithm to different inputs provided to the algorithm it was mandatory to include the developer's settings for making it accessible for users to debug any special case input errors.

WARNING: Enabling all Developer's modes shall exponentially slow down the algorithm and may cause it to exceed the maximum memory handling capabilities of hardware on which the algorithm is made to run.

```
%% Developer's Settings

dmode_1 = 0; 🡐 Disabling Developer's Mode 1
dmode_2 = 0; 🡐 Disabling Developer's Mode 2
```

Get the name of the object using "fileparts" algorithm.

```
%% Object to find
```

LOAD the OBJECT image and make 2 instances of it.

```
object_image = imread(object_path);
original_object_image = object_image;
```

Convert it to grayscale image

```
object_image = rgb2gray(object_image);

%% Displaying Object
```

If developer's mode enabled display the object image.

```
if dmode
SHOW the object image
imshow(org_img);
end
```

%% Scenery to find from

Similar to OBJECT, LOAD the SCENE image and make 2 instances of it further converting it to Grayscale.

%% Detection of SURF Features

```
object_SURF_features = detectSURFFeatures(object_image);
scene_SURF_features = detectSURFFeatures(scene_image);
```

If developer’s mode is enabled display the feature detected image.

%% GET Feature Descriptors

Retrieve the feature descriptors using the “extractFeatures” algorithm

```
[object_feature_descriptors, object_SURF_features] =
extractFeatures(object_image, object_SURF_features);

[scene_feature_descriptors, scene_SURF_features] =
extractFeatures(scene_image, scene_SURF_features);
```

%% Match the feature Descriptors

SURF Feature Descriptors are putatively matched and the matched descriptors are made stored in a separate array of points

```
[matched_feature_descriptors, matchmetric] =
matchFeatures(object_feature_descriptors, scene_feature_descriptors);
```

If developer’s mode is enabled display the matched feature descriptors detected in object scene image pair.

%% K-Means Machine Learning Algorithm

The value of “K” for K-Means ML Algorithm is manually set to increase efficiency of algorithm.

```
k = 10;

matchedScenePoints = scene_SURF_features;
```

Apply the K-Means Algorithm to make clusters of MATCHED SURF FEATURE DESCRIPTORS

```
idx = kmeans(matchedScenePoints.Location,k);
```

%% Algorithm Breakdown
Make further modifications to the “idx” matrix generated to rearrange the terms in the descending order of number of points present in the cluster

%% Plotting Clustered Points

Plot these clustered points on to the scene image with numbering in descending order of total number of cluster points present in one cluster. (Display of clustered points takes place fi developer’s mode 2 is enabled)

```
%% Min Eigen Boundary Detector
```

Detect the boundaries of the scene by using the Minimum eigenvalue feature detector.

```
scene_mev_features = detectMinEigenFeatures(sceneImage);

%% K-Means Machine Learning Algorithm for Minimum Eigenvalue features

k = 100;

matchedScenePoints = scene_mev_features;

idx_2 = kmeans(matchedScenePoints.Location,k);

%% Convex Hull Algorithm Call
```

Draw a polygon around the boundaries of the object in the scene such that all the matched points lie either on the polygon or inside it

x,y 🡐 X and Y coordinates of the selected matched Minimum Eigenvalue matched features

```
k = convhull (x,y); 🡐 Convex Hull Algorithm
imshow(original_scene_image);
plot(x(k),y(k),'r-',x,y,'b*')
```