The appendix includes some important data structures used in the 'C' programs that were developed to support the research work.

*CHAR_INFO*:

```c
struct CHAR_INFO{
    int v_segs;        //number of vertical segments across character
    int u_stroke;      //number of upper strokes in a character
    int m_loop;        //number of middle loops in a character
    int l_stroke;      //number of lower strokes in a character
};
```

The CHAR_INFO structure is used to store features for all alphabets. This helps in automatic generation of word dictionary, used for word recognition.

*DICT_WORD*:

```c
struct DICT_WORD {
    int no_of_chars;  //number of characters in a word
    char *word;       //the word itself
    int word_index;   //index of the word in word-dictionary
    int no_of_v_segs; //number of vertical segments in a word
    int no_of_upper_strokes; //number of upper strokes
    int no_of_middle_loops; //number of middle loops
    int no_of_lower_strokes; //number of lower strokes
    int *position_upper_strokes;//position of upper strokes
};
```
int *position_middle_loops; //position of middle loops
int *position_lower_strokes; //position of lower strokes
int no_of_instances; //number of instances of the word in total phrases
int *present_in_text_index; //Index of text in text dictionary, containing word
int *position_in_text; //and at which position in text string

};

The DICT_WORD structure stores entry for each word in 'word' dictionary.

* DICT_TEXT :

struct DICT_TEXT {
    int text_length; //length of phrase in terms of characters
    char *text; // the phrase itself
    int text_index; //index of phrase in text dictionary
    int no_of_words; //number of words in a phrase
    int *index_words; //indices of each phrase's word in word dictionary

};

The DICT_TEXT structure stores entry for each phrase in 'text' dictionary.

* H_FEATURE :

typedef struct tagH_FEATURE {
    RECT rect; //rectangle enclosing the feature
    int type; //type of higher feature
    int position; //fuzzy position
    int sizex, sizey; //maximum 10, max. sizex, sizey are width and height.
    int primitive_f_index1; //three or two primitive features which forms the
    int primitive_f_index2; //higher feature. Indices of primitive features in series
int primitive_f_index3; // index1, index2 and index3.

} H_FEATURE;

The H_FEATURE structure stores information regarding higher features. It also keeps indices of primitive features in a series, which form the higher feature.

* KB_STRUCT : 

typedef struct {
    float mean[SYMB_VEC_LEN];
    float variance[SYMB_VEC_LEN];
    int symbol_count;
} KB_STRUCT;

The KB_STRUCT is a knowledge base structure to store information regarding a character class. SYMB_VEC_LEN is the length of the vector which represent a character sample. 'mean' and 'variance' are the mean and variance vectors of a character class respectively. 'symbol_count' indicate the number of elements in the class from which mean and variance is computed.

* LOOP :

typedef struct tagLOOP {
    int xs, ys; // start point in the series
    int xe, ye; // and end point in the series which forms a loop
    RECT rect; // rectangle enclosing the loop
    int series1, series2; // xs, ys found in series1 and xe, ye found in series 2
    int insidelevel; // number of loops which covers this loop.
    int position; // fuzzy position
    int sizex, sizey; // maximum 10, max. sizex, sizey are width and height.
The LOOP structure stores information regarding a loop feature.

*GG*:

typedef struct tag_NNW{
int numb_input_nodes; //number of input nodes
int numb_hidden_nodes; //number of hidden nodes
int numb_output_nodes; //number of output nodes
float eta; //learning factor
float alpha; //momentum
float tolerable_err; //error in output which can be tolerated.
float input_range; //maximum input value, used to normalize input [0..1]
float output_range; //maximum output value, used to normalize output [0..1]
float weight_range; //range of initial weights, -weight_range to +weight_range
int flag_adjust_weight_at_epoch; //whether the weights are to be adjusted after
                                //each training sample or after an epoch.
float *p_input; //input values
float *p_hidden; //output values of hidden layer nodes
float *p_output; //output values of output layer nodes
float *p_desired; //desired output values
float bias_input; //value of bias for hidden nodes.
float bias_hidden; //value of bias for output nodes.
float *p_sum_hidden; //sigma values at hidden nodes
float *p_sum_output; //sigma values at output nodes
float *p_delta_weight_i2h; //delta weight_matrix between input and hidden
float *p_delta_weight_h2o; // delta weight matrix between hidden and output
float *p_delta_bias_weight_hidden; // delta bias weights of hidden nodes
float *p_delta_bias_weight_output; // delta bias weights of output nodes
float *p_weight_i2h; // weight matrix between input and hidden
float *p_weight_h2o; // weight matrix between hidden and output
float *p_bias_weight_hidden; // bias weights of hidden nodes.
float *p_bias_weight_output; // bias weights of output nodes.
float mean_square_err; // mean square error between output and desired output
float *p_bp_err_output; // errors at output nodes
float *p_bp_err_hidden; // errors at hidden nodes
float *p_bp_t_err_output_x_hidden; // sum of product of error output & hidden
float *p_bp_t_err_hidden_x_input; // sum of product of error hidden, and input
float *p_bp_t_err_output_x_bias_hidden; // sum of pro. of err. out., and b. hid.
float *p_bp_t_err_hidden_x_bias_input; // sum of pro. of err. hid., and b. inp.
}

The NNW structure stores structural information for a three-layered feedforward neural network: input, hidden and output layers. The structure is used by backpropagation algorithm to store training parameters, intermediate weights and delta weights, intermediate errors and delta errors and intermediate outputs.

* P_FEATURE :

typedef struct tagP_FEATURE {
    RECT rect; // rectangle enclosing the feature
    int type; // type of primitive feature
    int position; // fuzzy position
}
int sizex, sizey; //maximum 10, max. sizex, sizey are width and height.

} P_FEATURE;

The P_FEATURE structure stores information regarding a primitive feature.

* RECT :

typedef struct tagRECT {
    int xmin, ymin; //left top corner
    int xmax, ymax; //right bottom corner
} RECT;

The RECT structure store a rectangle’s information in form of diagonal points (xmin, ymin) and (xmax, ymax).

* SERIES :

typedef struct tagSERIES{
    RECT rect;      //rectangle enclosing the series
    int position;   //fuzzy position
    int sizex, sizey; //maximum 10, max. sizex, sizey are width and height
    int noofp_features; // number of primitive features in the series
    int noofh_features; // number of higher features in the series
    int formsloop;    // a flag indicating whether the series forms a loop
    P_FEATURE *pp_feature; // List of primitive features
    H_FEATURE *ph_feature; // List of higher features
} SERIES;

The SERIES structure stores information of a series obtain by tracing a contour image.
The STBFILEHEADER structure is the file header of *.stb files. Id1 and Id2 indicate that the image file is an .STB file. OffImg indicate the location of image contents, by byte-offset from the start of the file. The size of the image is:

$$\text{Size of the Image} = (\text{Height} \times \text{Width} / 8) \text{ bytes.}$$

and the size of the file is:

$$\text{Size of the File} = (\text{Size of the Image} + \text{OffImg}) \text{ bytes.}$$

The StbInfo header keeps the information regarding the image or any rectangular
portion of it in memory. The rectangular portion is given by diagonal points 
(XStart,YStart) and (XEnd,YEnd).

* SYMBOLATTRIB :

typedef struct tagSYMBOLATTRIB {
    int height;       // height of symbol
    int width;        // width of symbol
    int noofloops;    // number of loops in symbol
    LOOP *ploop;      // list of loops in symbol
    int noofseries;   // number of series in symbol contour
    SERIES *pseries;  // list of series in symbol
} SYMBOLATTRIB;

The SYMBOLATTRIB structures stores all attribute information obtained by 
processing a symbol image.

* UNKNOWN_WORD_INFO :

struct UNKNOWN_WORD_INFO{
    struct RECT whole_word; //rectangle enclosing word
    int height;             //height of word
    int width;              //width of word
    int *v_density;         //vertical density along the width of word
    int *h_density;         //horizontal density along the height of the word
    int reference_line_positions[4]; //top,top_middle,bottom_middle,bottom lines
    int no_of_v_segs;       //number of vertical segments cleaving word
    int *v_segs_pos;        //position of vertical segments
    int no_of_upper_strokes; //number of upper strokes
int no_of_lower_strokes; //number of lower strokes
int no_of_middle_loops; //number of middle loops
int *position_upper_strokes; //position of upper strokes
int *position_lower_strokes; //position of lower strokes
int *position_middle_loops; //position of middle loops
float *cv_upper_strokes; //confidence value of upper strokes
float *cv_middle_loops; //confidence value of middle loops
float *cv_lower_strokes; //confidence value of lower strokes
float cv_first_chars[NO_OF_CHARS]; //confidence value of reference
characters to be the first character.
float cv_match_words[NO_OF_MATCH_WORDS]; //confidence value of
closely matching words.
int match_words[NO_OF_MATCH_WORDS]; //indices of closely matching
words in word-dictionary.

};

The UNKNOWN.Word_INFO structure stores information regarding an unknown
word when it is being processed by text recognition program.

* UNKNOWN.TEXT_INFO :

struct UNKNOWN.TEXT_INFO{

struct RECT text_rect; //rectangle enclosing phrase
int height; //height of phrase
int width; //width of phrase
int *text_v_density; //vertical density along width of text
int no_of_words; //number of words found in text

263
The UNKNOWN_TEXTINFO structure stores information regarding an unknown phrase when it is being processed by text recognition algorithm.