Qualnet provides implementation of IEEE 802.16 standard in C++. To embed concepts of fuzzy logic in Qualnet implemented was done in C/C++. This appendix provides an overview of data structures and functionality required for some basic concepts used in fuzzy logic. Implementation of fuzzy logic requires storage/representation of following types of information.

- Inputs and outputs
- Input and output membership functions
- Antecedent values
- Rules
- Rule-output strengths.

A.1 Data Structure for Input /Output and membership functions

```c
struct mf_type{
    char name[5];
    float value;
    float p1,p2,s1,s2;
    mf_type *next;
};

struct io_type {
    char name[10];
}
```

Implementation of all these concepts is done by defining appropriate data structures used to hold their values. Inputs and outputs have been implemented by defining structure `io_type` and `mf_type` is defined for implementing membership functions. Code snippet A.1 shows code and various members for these structures. Structure `mf_type` has been declared as member of `io_type` so that every input or output variable can be associated with its respective membership function type.
Appendix A
Fuzzy Logic Implementation

Fig. A.1:- Data structures used for inputs and membership functions
The io_type structure is implemented as linked list with members as name, value, pointer of mf_type and a pointer to next input variable. Figure A.1 shows how input and membership functions are arranged. Structure mf_type consists of two X-axis points defined by p1 and p2 and two other two floating variables s1 and s2. These four points describe implementation of trapezoidal membership function as shown in figure A.2. This information is used to calculate antecedent values (degrees of membership) as explained later.

![Fig. A.1: Data structures used for inputs and membership functions](image)

Fig. A.2:- Implementation of trapezoidal and triangular membership functions
The value field in membership function data structure is used to store values of degree of membership or output strength. In order to define membership function values for four floating variables s1, s2, p1 and p2 will be inputted as shown in figure A.2. A triangular membership function can be defined by converting four points to three points as per same diagram.

p1 = point a
p2 = point d
s1 = \( m/(\text{point b- point a}) \)

s2 = \( m/(\text{point d - point c}) \)
struct rule_element
{
    float *value;
    rule_element *next;
};

struct rule_type
{
    rule_element *ifside;
    rule_element *thenside;
    rule_type *next;
};

rule_type *Rule_base = NULL;

Figure A.3 shows the linked list arrangement of rules used to implement fuzzy rule base. Such a structure can be implemented by defining two pointers to structure rule_element. The first pointer points to if-side of the rules and defines which values will be used to evaluate strength of rules using input variables. Second pointer points to output side where strength will be applied to calculate values for output variables. Both these pointers are grouped together in a single data structure rule_type which is
initially defined to NULL. Different rules in rules base are connected together by using pointer.

**A.2 Fuzzification:**- This function calculates degree of membership value for each membership of system input. The value so calculated corresponds to if-side of rules. It makes calls to another function known as degree_of_mship(...) to calculate degree of membership by passing membership function and pointer to system input value as argument. The function calculates extent of membership of input to corresponding membership function. Delta terms as explained in figure A.4 are calculated. If input is outside scope of membership function ie value of either of d1 or d2 <0, then zero is returned else smaller of d1*s1, d2*s2 and 1 is returned.

```c
fuzzification()
{
    io_type *si;     /* pointer pointing to input */
    mf_type *mf;
    //Loop for all inputs
    Repeat till all inputs are not NULL
    for(si=start_ip; si != NULL; si=si->next)
    {
        //Loop for all membership functions within each i/p
        for(mf=si->mem_func; mf != NULL; mf=mf->next)
            degree_of_mship(mf,si->value);
    }
}
```

![Fig. A.4:- Evaluation of degree of membership by computing delta terms](image-url)
A.3 Rule Evaluation:- Evaluation of rules is done by ANDING values for its if-sides together and applying a minimum function to evaluate rule strength. Once strength of rule is calculated it is applied to then part of output rules. For an already assigned output, conflict is resolved by applying maximum function at every intermediate pass.

```c
void evaluate_rules()
{
    Rule_type *rule;
    rule_element *ip; // if-side pointer
    rule_element *tp; // then side pointer
    float strength; // currently evaluated rule

    For all rules in rule-base
    {
        For every if-side of each rule
        {
            Calculate minimum of (strength, *(ip->value);
        }
        /* Apply strength calculated to then-side */
        For every then-side of the corresponding rule
        {
            calculate maximum of output value or strength
            *(tp->value) = max(strength, *(tp->value));
        }
    }
}
```

A.4 De-fuzzification: Last step of fuzzy inference, defuzzification, involves calculation of crisp value for our output variables. It is implemented using centre of gravity method. It is done by evaluating strengths of output rules and measuring areas for individual membership function using a function Area_Trapezoid(...). Sum of products and sum of shortened areas under output membership functions are evaluated using centroid of membership function and aggregating all individual areas together respectively. A final crisp values is obtained by dividing former by latter.

```c
void defuzzification()
{
    io_type *so; mf_type *mf;
    float SOP;
    float SOA;
```
float area;    // area of trapezium
float centroid;
Repeat for each output variable
{
    SOP = 0;    SOA = 0;
    //Loop for each membership function with in each variable
    {
        Calculate area of trapezoid(mf);
        Find centroind of membership function using
        x-axis points for mf.
        SOP += area * centroid;
        SOA += area;
    }
    so->value = SOP/SOA;
}
}
APPENDIX B
SAMPLE SCENARIO
PICTURES
Fig. B.1: 3-dimensional snapshot of WiMAX Scenario with 60 nodes in ζ
Fig. B.2: 3–dimensional snapshot of WiMAX Scenario with 75 nodes in Qualnet
Fig. B.3: 3–dimensional snapshot of WiMAX Scenario with 90 nodes in
Fig. B.4:- 3–dimensional snapshot of WiMAX Scenario with 105 nodes in Qual