**SORT CLASS**

```
//////////////////////////
// QUICKSORT.CPP
//////////////////////////

#include "qSort.h"
#include <windows.h>
#include <iostream.h>
#include <string.h>

/////////////////////////////////////////////////////////////////////
// values   : Array of values to sort
// start    : Index of the first element in the list to sort
// end_list : Index of the last element in the list to sort
// order    : Desired sorting order
//          0 -> ascending
//          1 -> descending
/////////////////////////////////////////////////////////////////////

//////////////////////////////////
// The Constructor Function
//////////////////////////////////
CqSort::CqSort()
{

}

//////////////////////////////////
// The Quick Sort Function
//////////////////////////////////
void CqSort::qSort(int* values,int start,int end_list,int order)
{
        int temp ;
        int low = start ;
        int high = end_list ;
        int list_separator = values[(start+end_list)/2];

        do
        {
                if (!order) // ascending
                {
                        while (values[low] < list_separator)
                                low++;
                        while (values[high] > list_separator)
```

```
                        high--;
                }else  //descending
                {
                        while (values[low] > list_separator)
                                low++;

                        while (values[high] < list_separator)
                                high--;
                }
                if (low <= high)
                {
                        temp = values[low];
                        values[low++] = values[high];
                        values[high--] = temp;
                }
        }
        while (low <= high);

                if (start < high)
                        qSort(values,start,high,order);
                if (low < end_list)
                        qSort(values,low,end_list,order);

}

void CqSort::qSort(char values[][200],int start,int end_list,int order)
{
        char temp [200];
        int low = start ;
        int high = end_list ;
        char list_separator[200];

        strcpy(list_separator, values[(start+end_list)/2]);

        do
        {
                if(!order) // ascending
                {
                        while(strcmpi(values[low],list_separator)<0)
                                low++;

                        while(strcmpi(values[high],list_separator)>0)
                                high--;
                }else // decending
                {
                        while (strcmpi(values[low],list_separator)>0)
                                low++;
                        while (strcmpi(values[high],list_separator)<0)
                                high--;
                }
```

```
                if(low <= high)
                {
                        strcpy(temp , values[low]);
                        strcpy(values[low++], values[high]);
                        strcpy(values[high--], temp);
                }
        }
        while (low <= high);

        if (start < high)
                qSort(values, start , high , order);
        if (low < end_list)
                qSort(values,low,end_list,order);
}

////////////////////////////////
// The Destructor Function
////////////////////////////////
CqSort::~CqSort()
{

}

//////////////////////
// QUICK SORT.H
//////////////////////

// Class declaration

class CqSort
{
public :
        CqSort(); // Constructor

        void qSort(int* values , int start , int end_list , int order);
        void qSort(char values[][200], int start , int end_list , int order);

        ~CqSort(); // Destructor
};

//////////////////////////
// SHELL SORT.CPP
//////////////////////////

#include "sSort.h"
#include <windows.h>
#include <iostream.h>
#include <string.h>
```

```
////////////////////////////////////////////////////////////
// values                  : Array of values to sort
// num_elements : number of elements in Array
// order          : Desired sorting order
//                                  0 -> ascending
//                                  1 -> descending
////////////////////////////////////////////////////////////


//////////////////////////////////
// The Constructor Function
//////////////////////////////////
CsSort::CsSort()
{

}

//////////////////////////////////
// The Shell Sort  Function
//////////////////////////////////
void CsSort::sSort(int* values, int num_elements, int order)
{

        int temp;
        int i,gap ,exchange_occurred;

        gap = num_elements/2 ;
        do
        do{
                exchange_occurred = 0;

                for(i=0; i<num_elements-gap; i++)
                        if((!order&&(values[i]>values[i+gap])) ||
                                (order && (values[i] < values[i+gap])))
                        {
                                temp = values[i];
                                values[i] = values[i+ gap];
                                values[i+gap] = temp ;
                                exchange_occurred = 1;
                        }
        }
        while (exchange_occurred);
        while (gap= gap/2);
}

void CsSort::sSort(char values[][200], int num_elements, int order)
{
        char temp [200];
        int i,gap,exchange_occurred;
        gap = num_elements/2;
```

```
        do
        do{
                exchange_occurred = 0;
                for(i=0;i<num_elements-gap;i++)
                if((!order&&(strcmpi(values[i],values[i+gap])>0)) ||
                        (order &&(strcmpi(values[i],values[i+gap]) <0)))
                        {
                                strcpy(temp,values[i]);
                                strcpy(values[i],values[i+gap]);
                                strcpy(values[i+gap],temp);
                                exchange_occurred = 1;
                        }
        }
        while(exchange_occurred);
        while(gap=gap/2);
}

////////////////////////////////
// The Destructor Function
////////////////////////////////
CsSort::~CsSort()
{

}

//////////////////////
// SHELL SORT.H
//////////////////////

// Class declaration

class CsSort
{
public :
        CsSort(); // Constructor
        void sSort(int* values, int num_elements, int order);
        void sSort(char values[][200], int num_elements, int order);

        ~CsSort(); // Destructor
};

////////////////////////////
// MERGE SORT.CPP
////////////////////////////

#include "mSort.h"
#include <windows.h>
#include <iostream.h>
#include <malloc.h>
```

```
////////////////////////////////////////////////
// first  : first element of Array1
// second       : first element of Array2
// third : last element of Array
////////////////////////////////////////////////

////////////////////////////////////
// The Constructor Function
////////////////////////////////////
CmSort::CmSort()
{

}

////////////////////////////////////
// The Merge Sort Function
////////////////////////////////////
void CmSort::mSort(int* k, int first , int second , int third)
{
        int i,j,l,*temp;

        i= first ; j= second ; l = 0;
        temp = (int*) malloc(sizeof k);

        while(i<second && j<= third)
        {
                if (k[i] <= k[j])
                {
                        temp[l] = k[i];
                        i++;
                }else
                {
                        temp[l] = k[j];
                        j++;
                }
                l++;
        }

        if(i>= second)
        {
                while (j<= third)
                {
                        temp[l] = k[j];
                        j++;
                        l++;
                }
        }else
        {
                while(i<second)
                {
```

```
                        temp[l] = k[i];
                        i++;
                        l++;
                }
        }

        for(i=0;i<l;i++)
                k[first+i] = temp[i];
}

////////////////////////////////
// The Destructor Function
////////////////////////////////
CmSort::~CmSort()
{

}

//////////////////////////
// MERGE SORT.H
//////////////////////////

// Class declaration

class CmSort
{
public :
        CmSort(); // Constructor

        void mSort(int* k, int first , int second , int third);

        ~CmSort(); // Destructor
};
```

**********

**INFERENCE ENGINE CLASS**

```
////////////////////////////////////
// FORWARD CHAINING.CPP
////////////////////////////////////

#include "Fward.h"
#include <windows.h>


////////////////////////////////
// The Constructor Function
////////////////////////////////
CFward::CFward() // Constructor
```

```
{

}

////////////////////////////////////////
// The Forward Chaining Function
////////////////////////////////////////
bool CFward::Fward(char fact[], char antec[])
{
        bool Temp= FALSE;

        if (fact = antec)
                Temp = TRUE ;

        return Temp ;

}


//////////////////////////////////
// The Destructor Function
//////////////////////////////////
CFward::~CFward() // Destructor
{

}

//////////////////////////////////
// FORWARD CHAINING.H
//////////////////////////////////


// Class declaration

class CFward
{
public :
        CFward(); // Constructor

        bool Fward(char fact[] , char antec[]);

        ~CFward(); // Destructor
};


////////////////////////////////////
// BACKWARD CHAINING.CPP
////////////////////////////////////

#include "Bward.h"
```

```
#include <windows.h>


////////////////////////////////////
// The Constructor Function
////////////////////////////////////
CBward::CBward() // Constructor
{

}

////////////////////////////////////////////
// The Backward Chaining Function
////////////////////////////////////////////
bool CBward::Bward(char fact[], char cond[])
{
        bool Temp= FALSE;

        if (fact = cond)
                Temp = TRUE ;

        return Temp ;

}

////////////////////////////////////
// The Destructor Function
////////////////////////////////////
CBward::~CBward() // Destructor
{

}

////////////////////////////////////
// BACKWARD CHAINING.H
////////////////////////////////////

// Class declaration


class CBward
{
public :
        CBward(); // Constructor
        bool Bward(char fact[], char cond[]);
        ~CBward(); // Destructor
};
```

**********

**CLASS FUZZY REASONING**

```
//////////////////////////
// FUZZYLOGIC.CPP
//////////////////////////

#include "Fuzlogic.h"
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
////////////////////////////////////
// The Constructor Function
////////////////////////////////////
CFuzlogic::CFuzlogic()
{

}


////////////////////////////////////
//  Set the value Function
////////////////////////////////////
void CFuzlogic::setval(float &h, float &l)
{
        highval=h;
        lowval=l;
}

////////////////////////////////////
// Set the low value Function
////////////////////////////////////
float CFuzlogic::getlowval()
{
        return lowval;
}

////////////////////////////////////
// Set the high value Function
////////////////////////////////////
float CFuzlogic::gethighval()
{
        return highval;
}

////////////////////////////////
// The getshare Function
////////////////////////////////
float CFuzlogic::getshare(const float & input)
{
```

```
// this member function returns the relative membership
// of an input , with a maximum of 1.0

float output;

if (input <= lowval)
        output = 0;
else
{
        if ((input > lowval)||(input < highval))
                        output = (input - lowval)/ (highval - lowval);
        else
        {
                if(input >= highval)
                        output = 1;
        };
};
return output ;
}

////////////////////////////////
// The Destructor Function
////////////////////////////////
CFuzlogic::~CFuzlogic()
{

}

//////////////////////
// FUZZY LOGIC.H
//////////////////////

// Class declaration


class CFuzlogic
{
private:
        float    lowval,highval;

public:
        CFuzlogic();  // Constructor

        void setval(float&,float&);
        float getlowval();
        float gethighval();
        float getshare(const float&);

        ~CFuzlogic();  // Destructor
};
```

```
//////////////////////////
// FUZZYSET.CPP
//////////////////////////

#include "FuzSet.h"
#include <windows.h>
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>


////////////////////////////////////
// The Constructor Function
////////////////////////////////////
CFuzset::CFuzset()
{

}

//////////////////////////////////
// The setname Function
//////////////////////////////////
void CFuzset::setname(char*n)
{
        strcpy(name,n);
}

//////////////////////////////////
// The getname Function
//////////////////////////////////
char* CFuzset::getname()
{
        return name;
}

////////////////////////////
// The setval Function
////////////////////////////
void CFuzset::setval(float &h, float &m, float &l)
{
        highval = h;
        midval  = m;
        lowval  = l;
}

//////////////////////////////
// The getlowval Function
//////////////////////////////
float CFuzset::getlowval()
```

```
{
        return lowval;
}

////////////////////////////////
// The getmidval Function
////////////////////////////////
float CFuzset::getmidval()
{
        return midval;
}

////////////////////////////////
// The gethighval Function
////////////////////////////////
float CFuzset::gethighval()
{
        return highval;
}

////////////////////////////////
// The getshare Function
////////////////////////////////
float CFuzset::getshare(const float & input)
{
        float output ;
        float midlow , highmid ;

        midlow  = midval  - lowval ;
        highmid = highval - midval ;

        // if outside the range. then output = 0
        if((input <= lowval )||(input >= highval))
                output = 0;
        else
        {
                if(input > midval)
                        output = (highval - input)/highmid;
                else
                        if(input == midval)
                                output = 1.0;
                        else
                                output = (input - lowval)/midlow;
        }
        return output ;
}
```

```
#include <windows.h>

////////////////////////////////////
// The Constructor Function
////////////////////////////////////
CBestFirst::CBestFirst()
{

}

//////////////////////////////////////////
// The BestFirst Search Function
//////////////////////////////////////////
CBestFirst::Best_First(int Index , int value_right , int value_left)
{

        if(value_right >= value_left)
                return Index * 2;
        else
                return Index *2 + 1;

}

////////////////////////////////
// The Destructor Function
////////////////////////////////
CBestFirst::~CBestFirst()
{

}

//////////////////////////////
// BEST FIRST SEARCH.H
//////////////////////////////

// Class declaration

class CBestFirst
{
public :
        CBestFirst(); // Constructor

        int Best_First(int Index , int value_right , int value_left);

        ~CBestFirst(); // Destructor
};
```

**********

**CLASS NPL**

```
////////////////////////////////////
// SYNTAX ANALYSIS .CPP
////////////////////////////////////

#include "SynAnalysis.h"
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

//////////////////////////
// Separator Data
//////////////////////////
void CSyntax::seps_Data(char* value[], char list[])
{
        char* token ;
        char seps[]= " ";// Find space
        int i = 0;

        //LPTSTR lpsz = new TCHAR[list.GetLength()+1];
        //strcpy(lpsz , list);

        token = strtok(list , seps);
        while (token != NULL)
        {
                //LPTSTR lpsz = new TCHAR[value.GetLength()+1];
                strcpy(value[i], token);
                i++;
                token = strtok(NULL , seps);
        }
}

////////////////////////////////
// Syntax Analysis Data
////////////////////////////////

// analy[]: syntax analysis data
void Syntax_Analysis(char* analy[] , char data[] , int Index)
{
        strcpy(analy[Index],data);
}

//////////////////////////////
// SYNTAX ANALYSIS.H
//////////////////////////////

// Class declaration
```

```
//////////////////////////////////////////
// seps_Data function
// list  : input
// value : array of separate input
//////////////////////////////////////////

class CSyntax
{
public:
        CSyntax() {} ;

        void seps_Data(char* value[], char list[]);
        void Syntax_Analysis(char* , char data[] , int Index);

        ~CSyntax() {} ;
};

////////////////////////////////////
// SEMENTIC ANALYSIS.CPP
////////////////////////////////////

#include "SemAnaly.h"
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

//////////////////////////////////////////////////////////////////////
// Sem_Analysis
// analy[] : S -> NP,VP
// Semantic [] : show semantic analysis of sentence
//////////////////////////////////////////////////////////////////////

void CSemantic::Sem_Analysis(char* analy[] ,char semantic[][100])
{
        char suffix1[] = " is a Subject.";
        char prefix1[] = "The Subjec is  ";
        char suffix2[] = " is a object.";

        if ((analy[0] = "DET") || (analy [1] = "N"))
        {
                strcpy(semantic[1],analy[1]);
                strncat( semantic[1], suffix1, 20 );
        }
        if ((analy[1] = "N")||(analy[2] = "V"))
        {
                strcpy(semantic[2],analy[2]);
                strncat( prefix1,semantic[1], 20 );
        }
        if ((analy[1] = "N") || (analy[2] = "V") || (analy[3] = "N"))
```

```
        {
                strcpy(semantic[3],analy[3]);
                strncat( semantic[3], suffix2, 20 );
        }
}

////////////////////////////////////
// SEMANTIC ANALYSIS.H
////////////////////////////////////

// Class declaration


class CSemantic
{
public:
        CSemantic() {} ;

        void Sem_Analysis(char* analy[] ,char semantic[][100]);

        ~CSemantic() {} ;
};
```

**********

# ประวัติผู้เขียน

**ชื่อ** นายนพดล เศรษฐ์ชัยยันต์

**วัน เดือน ปี เกิด** 10 กุมภาพันธ์ 2509

**ประวัติการศึกษา**

- ครุศาสตร์อุตสาหกรรมบัณฑิต สาขาวิชาอิเล็กทรอนิกส์ คณะวิศวกรรมเทคโนโลยี สถาบันเทคโนโลยีราชมงคล วิทยาเขตภาคพายัพ ปีการศึกษา 2536