APPENDIX

WATER CLASSES AND USES

Kelas *Class*	**Kegunaan** *Uses*
Kelas I *Class I*	Pemuliharaan alam semula jadi *Conservation of natural environment.*
	Bekalan Air I – Hampir tiada rawatan diperlukan. *Water Supply I – Practically no treatment necessary.*
	Perikanan I – Spesis akuatik yang sangat sensitif. *Fishery I – Very sensitive aquatic species.*
Kelas IIA *Class IIA*	Bekalan Air II – Memerlukan rawatan secara konvensional sahaja. *Water Supply II – Conventional treatment required.*
	Perikanan II – Spesis akuatik yang sensitif. *Fishery II – Sensitive aquatic species.*
Kelas IIB Class IIB	Kegunaan rekreasi yang melibatkan persentuhan badan dengan air. *Recreational use with body contact.*
Kelas III *Class III*	Bekalan Air III – Memerlukan rawatan yang ekstensif. *Water Supply III – Extensive treatment required.*
	Perikanan III – Spesis tertentu yang mempunyai nilai ekonomi biasa. Bekalan air minum haiwan ternakan. *Fishery III – Common, of economic value and tolerant species.* *Livestock drinking.*
Kelas IV *Class IV*	Pengairan *Irrigation*
Kelas V *Class V*	Tiada seperti di atas. *None of the above.*

INTERIM NATIONAL WATER QUALITY STANDARDS FOR MALAYSIA

CLASSES							
PARAMETERS	UNIT	I	IIA	IIB	III	IV	V
Ammoniacal Nitrogen	mg/l	0.1	0.3	0.3	0.9	2.7	>2.7
BOD	mg/l	1	3	3	6	12	>12
COD	mg/l	10	25	25	50	100	>100
DO	mg/l	7	5 - 7	5 - 7	3 - 5	<3	<1
pH		6.5 - 8.5	6 - 9	6 - 9	5 - 9	5 - 9	-
Colour	TCU	15	150	150	-	-	-
Elec. Conductivity *	umhos/cm	1000	1000	-	-	6000	-
Floatables		N	N	N	-	-	-
Odour		N	N	N	-	-	-
Salinity (%)	%	0.5	1	-	-	2	-
Taste		N	N	N	-	-	-
Total Dissolved Solid	mg/l	500	1000	-	-	4000	-
Total Suspended Solid	mg/l	25	50	50	150	300	300
Temperature (C)	^{o}C	-	Normal $+2^{0}C$		Normal $+2^{0}C$	-	-
Turbidity (NTU)	NTU	5	50	50	-	-	-
Faecal Coliform **	counts/100mL	10	100	400	5000 (20000) a	5000 (20000) a	-
Total Coliform	counts/100mL	100	5000	5000	50000	50000	>50000

C SCRIPT FOR HEA TRAINING
Variable Declaration and Main Programme


```
/*
 * newbootstrapevolvesinglerule.c 15/09/2007

 * Copyright (c) 2007 School of Earth and Environmental Science.
 * The University of Adelaide, SA 5005 AUSTRALIA.
 * All rights reserved.
 */

/* @version 1.00
 * @author Hongqing Cao
 */

#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

char filename[50];
int *ndepth1,*ndepth2,opeone,size,opetwo,ai=0,bi=100;
int runtime;
time_t t,start,end;
float min_q,preerror,fiterror;
float pmax=500;

typedef unsigned char byte;
typedef struct
{
  int type;
  int index;
  int flag;
} ope_type;

typedef struct
{
  int type1;
  int type2;
  int index;
  float value;
} node_type;


int stopnum,depth1,depth2,elements1,elements2,seeds,datas,generation,step,numx;
float **datax,*datay,*tempdatay1,*tempdatay2;
float *rulesetdataf,*dataf,*tempdataf1,*tempdataf2;
char *logi_op_char[]={ "AND","OR"};
```

```
char *comp_op_char[]={ ">","<",">=","<="};
char *algo_op_char[]={ "+","-","*","/","sin","cos","exp","ln"};

char string[60000],*str;
float *q,totalq;
node_type ****s;
ope_type ope[8];
int line, addline;

float pr,pc,pm;
node_type ****news;
float   *newq;
int *fitness;
float ****oldparapop;
float *oldparafit;
int **numc;
int maxsize=1;
node_type ***tempruleset;
int *newrnum, *rnum, *bestrunrnum;
int *varflag;
int *totalcases;
float r2[2];
char **xvarname, *yvarname;

float totalr,totalerror;
float **origindatax,*origindatay;
float *origindataf;
float *tempdatay,*tempdataf;

float trainpercent,bestpercent;
int runs,runcount;
int bestnum;
node_type ****bestruns;
float   *bestrunq;

void mainprocedure();
float randf();
void readdata();
void create_rulesetpop();
void bub_sort_rulesetpop();
void xchg_ruleset();
void xchg_tree();
int checkc();
float randfab();
void selectoperation();
int computeindex();
int simplify_ruleset();
int checkcondition();
void normalize_ruleset();
void simplify_algo_tree();
```

```
void simplify_comp_tree();
void simplify_logi_tree();
void del_subtree1();
void move_subtree1();
void normalize_algo_tree();
int create_ruleset();
void build_ruleiftree2();
void build_tree2();
void build_tree1();
char* print_tree2();
void print_subtree2();
char* print_tree1();
void print_subtree1();
float cal_ruleset_fitness();
float cal_ruleset_predicterror();
int cal_ruleset_ifno();
float cal_ruleset_conclusion();
float cal_subtree1();
float cal_subtree2();
float get_value();
int randi();
void ruleset_parameter_optimization();
void init_ruleset_parapop();
float cal_ruleset_parafitness();
void sort_ruleset_parapop();
int same_ruleset();
int tournament();
void ruleset_elitism();
void crossover_tree1();
void crossover_tree2();
int crossover_ruleset();
void mutation_tree1();
void mutation_tree2();
int cal_tree1_nodes();
int cal_tree2_nodes();
int cal_tree1_highth();
int cal_tree2_highth();
int mapping_tree1();
int mapping_tree2();
void count_variable();
void count_totalcases();
float cal_r2();
void bootstrapping();
main(argc, argv)
int     argc;
```

```
char    *argv[];
{
        argc=2;
        //strcpy(filename,argv[1]);
        mainprocedure();
}


void mainprocedure()
{
        int rant,ran,nn,n,i,j,jj,ii,k,m,n1,n2,n3,sno,iii;
        FILE *fp;
        node_type *temptree1,*temptree2;
        float ranf;
        int ifno;
        char str1[300],st[30000],title[60000],fname[50];
        int *optimized,nopti,same;
        float *crate,*mrate;
        float e,x;

        depth1=4;
        depth2=4;
        elements1=1<<(depth1+1);
        elements2=1<<(depth2+1);
        stopnum=100;
        strcpy(filename,"testdata.txt");
        seeds=200;
        for(i=0;i<8;i++)
                ope[i].flag=1;
        for(i=4;i<6;i++)
                ope[i].flag=0;
        str=(char *)malloc(60000*sizeof(char));
        if(str==NULL)
        {
                printf("Out of memory!\n");
                return;
        }
        t=time(NULL);
        time(&start);
        srand(t);
        readdata();
        bestnum=(int)(runs*bestpercent);
        if(bestnum==0) bestnum=1;

    bestrunq=(float *)malloc(runs*sizeof(float));
        bestruns=(node_type ****)malloc(runs*sizeof(node_type ***));
        bestrunrnum=(int *)malloc(runs*sizeof(int));

        for(i=0;i<runs;i++)
                bestruns[i]=(node_type ***) malloc(2*sizeof(node_type **));
```

```
for(i=0;i<runs;i++)
{
        bestruns[i][0]=(node_type **)malloc(maxsize*sizeof(node_type *));
        bestruns[i][1]=(node_type **)malloc((maxsize+1)*sizeof(node_type *));
}
for(i=0;i<runs;i++)
        for(j=0;j<maxsize;j++)
                bestruns[i][0][j]=(node_type*)malloc(elements1*sizeof(node_type));
for(i=0;i<runs;i++)
        for(j=0;j<maxsize+1;j++)
                bestruns[i][1][j]=(node_type*)malloc(elements2*sizeof(node_type));

totalcases=(int *)malloc(maxsize*sizeof(int));
tempruleset=(node_type ***)malloc(2*sizeof(node_type **));
tempruleset[0]=(node_type **) malloc(maxsize*sizeof(node_type *));
tempruleset[1]=(node_type **) malloc((maxsize+1)*sizeof(node_type *));
for(i=0;i<maxsize;i++)
        tempruleset[0][i]=(node_type*)malloc(elements1*sizeof(node_type));
for(i=0;i<maxsize+1;i++)
        tempruleset[1][i]=(node_type*)malloc(elements2*sizeof(node_type));
newq=(float *)malloc(seeds*sizeof(float));
news=(node_type ****)malloc(seeds*sizeof(node_type ***));
newrnum=(int *)malloc(seeds*sizeof(int));

for(i=0;i<seeds;i++)
        news[i]=(node_type ***) malloc(2*sizeof(node_type **));
for(i=0;i<seeds;i++)
{
        news[i][0]=(node_type **)malloc(maxsize*sizeof(node_type *));
        news[i][1]=(node_type **)malloc((maxsize+1)*sizeof(node_type *));
}
for(i=0;i<seeds;i++)
        for(j=0;j<maxsize;j++)
                news[i][0][j]=(node_type*)malloc(elements1*sizeof(node_type));
for(i=0;i<seeds;i++)
        for(j=0;j<maxsize+1;j++)
                news[i][1][j]=(node_type*)malloc(elements2*sizeof(node_type));
temptree1=(node_type*)malloc(elements1*sizeof(node_type));
temptree2=(node_type*)malloc(elements2*sizeof(node_type));
varflag=(int *)malloc(numx*sizeof(int));
crate=(float *)malloc(seeds*sizeof(float));
mrate=(float *)malloc(seeds*sizeof(float));
ndepth1=(int *) malloc(elements1*sizeof(int));
ndepth2=(int *) malloc(elements2*sizeof(int));
optimized=(int *)malloc(seeds*sizeof(int));
dataf=(float *)malloc((line+addline)*sizeof(float));
origindataf=(float *)malloc(datas*sizeof(float));
tempdatay=(float *)malloc(datas*sizeof(float));
tempdataf=(float *)malloc(datas*sizeof(float));
tempdatay1=(float *)malloc((line+addline)*sizeof(float));
```

```
	tempdataf1=(float *)malloc((line+addline)*sizeof(float));
	tempdatay2=(float *)malloc((line+addline)*sizeof(float));
	tempdataf2=(float *)malloc((line+addline)*sizeof(float));
	numc=(int **)malloc(2*sizeof(int *));
	numc[0]=(int *)malloc(maxsize*sizeof(int));
	numc[1]=(int *)malloc((maxsize+1)*sizeof(int));
	k=1;
	for(i=0;i<=depth1;i++)
	{
		for(j=k;j<2*k;j++)
			ndepth1[j]=i;
		k=k*2;
	}
	k=1;
	for(i=0;i<=depth2;i++)
	{
		for(j=k;j<2*k;j++)
			ndepth2[j]=i;
		k=k*2;
	}
if((fp=fopen("result.txt","wt"))==NULL)
	{
		printf("Cannot open output file !\n");
		exit(1);
	}
fclose(fp);
if((fp=fopen("variable.txt","wt"))==NULL)
	{
		printf("Cannot open output file !\n");
		exit(1);
	}
fclose(fp);

for(runcount=0;runcount<runs;runcount++)
{
	step=0;
	generation=0;
	bootstrapping(line);
	printf("Runno:%d depth1=%d depth2=%d popsize=%d totalline=%d line=%d
addline=%d \n",runcount,depth1,depth2,seeds,datas,line,addline);
	create_rulesetpop();
	while(generation<stopnum)
   {
		for (i=0;i<seeds;i++)
		{
			rnum[i]=simplify_ruleset(s[i],rnum[i]);
			normalize_ruleset(s[i],rnum[i]);
		}
		for(ii=0; ii<seeds; ii++)
		{
```

```
                sno=-1;
                same=0;
                if(ii==0)
                {
                        sno=0;
                        optimized[0]=0;
                        nopti=1;
                }
                else
                {
                        for(jj=0; jj<nopti; jj++)
                        {

  if(same_ruleset(s[ii],s[optimized[jj]],rnum[ii],rnum[optimized[jj]]))
                                {
                                        same=1;
                                        break;
                                }
                        }
                        if(same==0)
                        {
                                sno=ii;
                                optimized[nopti]=ii;
                                nopti++;
                        }
                }
                if(sno!=-1)
                {

                        ruleset_parameter_optimization(sno);
                }
  }
        bub_sort_rulesetpop(s,q,seeds,rnum);
        for(i=0;i<seeds;i++)
        {
                nn=tournament(4);
      for(jj=0;jj<rnum[nn];jj++)
                        for(m=0;m<elements1;m++)
                                news[i][0][jj][m]=s[nn][0][jj][m];
                for(jj=0;jj<rnum[nn]+1;jj++)
                        for(m=0;m<elements2;m++)
                                news[i][1][jj][m]=s[nn][1][jj][m];
  newq[i]=q[nn];
                newrnum[i]=rnum[nn];
        }
ruleset_elitism();
        for(i=0;i<seeds; i++)
        {
                for(jj=0;jj<newrnum[i];jj++)
                        for(j=0;j<elements1;j++)
```

```
                        s[i][0][jj][j]=news[i][0][jj][j];
            for(jj=0;jj<newrnum[i]+1;jj++)
                for(j=0;j<elements2;j++)
                        s[i][1][jj][j]=news[i][1][jj][j];
            q[i]=newq[i];
            rnum[i]=newrnum[i];
        }
        bub_sort_rulesetpop(s,q,seeds,rnum);
        for(i=0;i<seeds;i++)
        {
            mrate[i]=0.3*(1-q[seeds-1]/q[i]);
            crate[i]=0.9-mrate[i];
        }
        for(i=0;i<seeds;i++)
        {
            j=randi(seeds);
            k=randi(seeds);
            while(j==k)
            {
                j=randi(seeds);
                k=randi(seeds);
            }
            if(q[j]<=q[k])
            {
                pc=crate[j];
                pm=mrate[j];
            }
            else
            {
                pc=crate[k];
                pm=mrate[k];
            }
            ranf=randf();
            if(ranf<pc)
            {
                ran=randi(2);
                if(ran==0)
                {


newrnum[i]=crossover_ruleset(s[j],s[k],rnum[j],rnum[k],tempruleset);
                        for(jj=0;jj<newrnum[i];jj++)
                            for(n=0;n<elements1;n++)
                                news[i][0][jj][n]=tempruleset[0][jj][n];
                        for(jj=0;jj<newrnum[i]+1;jj++)
                            for(n=0;n<elements2;n++)
                                news[i][1][jj][n]=tempruleset[1][jj][n];
                }
                else
                {
```

```
                ran=randi(2);
                nn=ran==0? j:k;
                for(jj=0;jj<rnum[nn];jj++)
                        for(n=0;n<elements1;n++)
                                news[i][0][jj][n]=s[nn][0][jj][n];
                for(jj=0;jj<rnum[nn]+1;jj++)
                        for(n=0;n<elements2;n++)
                                news[i][1][jj][n]=s[nn][1][jj][n];
                n1=randi(rnum[j]);
                n2=randi(rnum[k]);
                if(nn==j)
                        n3=n1;
                else n3=n2;
                crossover_tree1(s[j][0][n1],s[k][0][n2],temptree1);
                for(n=0;n<elements1;n++)
                        news[i][0][n3][n]=temptree1[n];
                n1=randi(rnum[j]+1);
                n2=randi(rnum[k]+1);
                if(nn==j)
                        n3=n1;
                else n3=n2;
                crossover_tree2(s[j][1][n1],s[k][1][n2],temptree2);
                for(n=0;n<elements2;n++)
                        news[i][1][n3][n]=temptree2[n];
                newrnum[i]=rnum[nn];
        }
}
else if(ranf<(pc+pm))
{
        ran=randi(2);
        nn=ran==0? j:k;
        for(jj=0;jj<rnum[nn];jj++)
                for(n=0;n<elements1;n++)
                        news[i][0][jj][n]=s[nn][0][jj][n];
        for(jj=0;jj<rnum[nn]+1;jj++)
                for(n=0;n<elements2;n++)
                        news[i][1][jj][n]=s[nn][1][jj][n];
        rant=randi(rnum[nn]);
        mutation_tree1(s[nn][0][rant],temptree1);
        for(n=0;n<elements1;n++)
                news[i][0][rant][n]=temptree1[n];
        rant=randi(rnum[nn]+1);
        mutation_tree2(s[nn][1][rant],temptree2);
        for(n=0;n<elements2;n++)
                news[i][1][rant][n]=temptree2[n];
        newrnum[i]=rnum[nn];
}
else
{
        ran=randi(2);
```

```
                        nn=ran==0? j:k;
                        for(jj=0;jj<rnum[nn];jj++)
                                for(n=0;n<elements1;n++)
                                        news[i][0][jj][n]=s[nn][0][jj][n];
                        for(jj=0;jj<rnum[nn]+1;jj++)
                                for(n=0;n<elements2;n++)
                                        news[i][1][jj][n]=s[nn][1][jj][n];
                        newrnum[i]=rnum[nn];
                }
        }
        for(i=0;i<seeds;i++)
                newq[i]=cal_ruleset_fitness(news[i],newrnum[i]);
        for(i=0;i<seeds;i++)
        {
                while(checkcondition(news[i],newrnum[i])==1)
                {
                        newrnum[i]=create_ruleset(news[i]);
                        newq[i]=cal_ruleset_fitness(news[i],newrnum[i]);
                        newrnum[i]=simplify_ruleset(news[i],newrnum[i]);
                        normalize_ruleset(news[i],newrnum[i]);
                }
        }
        ruleset_elitism();
        bub_sort_rulesetpop(news,newq,seeds,newrnum);
        for(i=0;i<seeds;i++)
        {
                newrnum[i]=simplify_ruleset(news[i],newrnum[i]);
                normalize_ruleset(news[i],newrnum[i]);
        }
        i=0;
        while(i<(seeds-1))
        {
                if(same_ruleset(news[i],news[i+1],newrnum[i],newrnum[i+1]))
                {
                        j=1;

while(same_ruleset(news[i],news[i+j],newrnum[i],newrnum[i+j]))
                        {
                                j++;
                                if((i+j)>(seeds-1)) break;
                        }
                        j--;
                        for(jj=0;jj<newrnum[i+j];jj++)
                                for(n=0;n<elements1;n++)
                                        news[i][0][jj][n]=news[i+j][0][jj][n];
                        for(jj=0;jj<newrnum[i+j]+1;jj++)
                                for(n=0;n<elements2;n++)
                                        news[i][1][jj][n]=news[i+j][1][jj][n];
                        newq[i]=newq[i+j];
                        newrnum[i]=newrnum[i+j];
```

```
                        for(k=i+1;k<=i+j;k++)
                        {
                                rant=randi(newrnum[k]+1);
                                mutation_tree2(news[k][1][rant],temptree2);
                                for(n=0;n<elements2;n++)
                                        news[k][1][rant][n]=temptree2[n];
                                newrnum[k]=simplify_ruleset(news[k],newrnum[k]);
                                newq[k]=cal_ruleset_fitness(news[k],newrnum[k]);
                        }
                        i=i+j;
                }
                else i++;
        }
        for(i=0;i<seeds; i++)
        {
                for(jj=0;jj<newrnum[i];jj++)
                        for(j=0;j<elements1;j++)
                                s[i][0][jj][j]=news[i][0][jj][j];
                for(jj=0;jj<newrnum[i]+1;jj++)
                        for(j=0;j<elements2;j++)
                                s[i][1][jj][j]=news[i][1][jj][j];
                q[i]=newq[i];
                rnum[i]=newrnum[i];
        }
        bub_sort_rulesetpop(s,q,seeds,rnum);
        for(i=0;i<seeds;i++)
        {
                rnum[i]=simplify_ruleset(s[i],rnum[i]);
                normalize_ruleset(s[i],rnum[i]);
        }
        if(q[seeds-1]!=min_q)
        {
                min_q=q[seeds-1];
                step=0;
        }
        else  step++;
        time(&end);
        runtime=difftime(end,start);
        preerror=cal_ruleset_predicterror(s[seeds-1],rnum[seeds-1]);

    printf("\n\n\n\n*********************************************\nRunNO:    %d
Generation:%d   runtime:%d\n",runcount,generation,runtime);
        printf("\n\nThe best ruleset is:\n ");
        for(i=0;i<rnum[seeds-1];i++)
        {
                printf(" IF %s\n",print_tree1(s[seeds-1][0][i]));
                printf(" THEN f(x)=%s\n",print_tree2(s[seeds-1][1][i]));
                printf(" ELSE \n");
        }
        printf(" f(x)=%s\n",print_tree2(s[seeds-1][1][rnum[seeds-1]]));
```

```
		printf("\nfitting error=%.4f  predicted error=%.4f ",q[seeds-1],preerror);
		generation++;
		}/* end of each run*/
		for(jj=0;jj<rnum[seeds-1];jj++)
				for(j=0;j<elements1;j++)
					bestruns[runcount][0][jj][j]=s[seeds-1][0][jj][j];
		for(jj=0;jj<rnum[seeds-1]+1;jj++)
				for(j=0;j<elements2;j++)
					bestruns[runcount][1][jj][j]=s[seeds-1][1][jj][j];
		for(i=0;i<datas;i++)
		{
			ifno=cal_ruleset_ifno(s[seeds-1],origindatax[i],rnum[seeds-1]);
			origindataf[i]=cal_ruleset_conclusion(ifno,s[seeds-
1],origindatax[i],rnum[seeds-1]);
		}
		e=0;
		for(i=0;i<datas;i++)
		{
			x=origindatay[i]-origindataf[i];
			if(fabs(x)<1.0e+5) e+=x*x;
			else
				e=1.0e+10;
		}
		totalerror=sqrt(e/datas);
		bestrunq[runcount]=totalerror;
		bestrunrnum[runcount]=rnum[seeds-1];
		}
	bub_sort_rulesetpop(bestruns,bestrunq,runs,bestrunrnum);
	time(&end);
	runtime=difftime(end,start);
	for(iii=0;iii<bestnum;iii++)
	{

		sprintf(str1,"_best%d",iii+1);
  for(i=0;i<line+addline;i++)
	{
		ifno=cal_ruleset_ifno(bestruns[runs-1-iii],datax[i],bestrunrnum[runs-1-iii]);
		dataf[i]=cal_ruleset_conclusion(ifno,bestruns[runs-1-
iii],datax[i],bestrunrnum[runs-1-iii]);
	}
	for(i=0;i<line;i++)
	{
	tempdatay1[i]=datay[i];
	tempdataf1[i]=dataf[i];
	}
	for(i=0;i<addline;i++)
	{
	tempdatay2[i]=datay[line+i];
	tempdataf2[i]=dataf[line+i];
	}
```

```
   r2[0]=cal_r2(tempdatay1,tempdataf1,line);
        r2[1]=cal_r2(tempdatay2,tempdataf2,addline);
        fiterror=cal_ruleset_fitness(bestruns[runs-1-iii],bestrunrnum[runs-1-iii]);
        preerror=cal_ruleset_predicterror(bestruns[runs-1-iii],bestrunrnum[runs-1-iii]);
        count_variable(bestruns[runs-1-iii],bestrunrnum[runs-1-iii]);
        for(i=0;i<datas;i++)
        {
                ifno=cal_ruleset_ifno(bestruns[runs-1-iii],origindatax[i],bestrunrnum[runs-1-
iii]);
                origindataf[i]=cal_ruleset_conclusion(ifno,bestruns[runs-1-
iii],origindatax[i],bestrunrnum[runs-1-iii]);
        }
        for(i=0;i<datas;i++)
        {
        tempdatay[i]=origindatay[i];
        tempdataf[i]=origindataf[i];
        }
        totalr=cal_r2(tempdatay,tempdataf,datas);
        e=0;
        for(i=0;i<datas;i++)
        {
                x=origindatay[i]-origindataf[i];
                if(fabs(x)<1.0e+5) e+=x*x;
                else
                        e=1.0e+10;
        }
        totalerror=sqrt(e/datas);

        if((fp=fopen("result.txt","at"))==NULL)
        {
                printf("Cannot open output file !\n");
                exit(1);
        }
        fprintf(fp,"%f   %.2f  \n",totalerror,totalr);
        fclose(fp);
        if((fp=fopen("variable.txt","at"))==NULL)
        {
                printf("Cannot open output file !\n");
                exit(1);
        }
        for(i=0;i<numx;i++)
                fprintf(fp,"%d  ",varflag[i]);
        strcpy(st,"\n");
        fprintf(fp,"%s",st);
        fclose(fp);

        strcpy(fname,"validationresult");
        strcat(fname,str1);
        strcat(fname,".txt");
```

```
	if((fp=fopen(fname,"wt"))==NULL)
	{
		printf("Cannot open output file !\n");
		exit(1);
	}
	for(i=0;i<numx;i++)
		fprintf(fp,"%s    ",xvarname[i]);
	fprintf(fp,"%s     predicted_%s\n",yvarname,yvarname);
	for(i=0;i<line+addline;i++)
	{
		title[0]='\0';
		for(j=0; j<numx; j++)
		{
			fprintf(fp,"%8.3f ",origindatax[i][j]);
		}
		fprintf(fp,"%8.3f   %8.3f \n",origindatay[i], origindataf[i]);
	}
	fclose(fp);
	strcpy(fname,"lastmodel");
	strcat(fname,str1);
	strcat(fname,".txt");
	if((fp=fopen(fname,"wt"))==NULL)
	{
		printf("Cannot open output file !\n");
		exit(1);
	}
	strcpy(st,"\n********Final Result Report********\n");
	fprintf(fp,"%s",st);
	strcpy(st,"Parameter Setting:\n");
	fprintf(fp,"%s",st);
	fprintf(fp,"Data File: %s\n",filename);
	strcpy(st,"Function Set:\n");
	fprintf(fp,"%s",st);
	fprintf(fp," +:%d -:%d *:%d /:%d ",ope[0].flag,ope[1].flag,ope[2].flag,ope[3].flag);
	fprintf(fp,"          sin:%d          cos:%d          exp:%d
ln:%d\n",ope[4].flag,ope[5].flag,ope[6].flag,ope[7].flag);
	fprintf(fp,"ruleset size= %d  if tree Max depth: %d  then-else tree Max depth: %d
Popsize: %d  Max Geno: %d  \n\n",maxsize,depth1,depth2,seeds,stopnum);
		strcpy(st,"\nThe best ruleset is:\n");
	fprintf(fp,"%s",st);
	for(i=0;i<bestrunrnum[runs-1-iii];i++)
	{
		fprintf(fp," IF %s\n",print_tree1(bestruns[runs-1-iii][0][i]));
		fprintf(fp," THEN %s=",yvarname);
		fprintf(fp,"%s\n",print_tree2(bestruns[runs-1-iii][1][i]));
		fprintf(fp," ELSE \n");
	}
	fprintf(fp," %s=",yvarname);
	fprintf(fp,"%s\n",print_tree2(bestruns[runs-1-iii][1][bestrunrnum[runs-1-iii]]));
	fprintf(fp,"\ntotal error=%.4f  total R2 value=%.2f\n ",totalerror,totalr);
```

```
        fclose(fp);
}
for (i=0;i<seeds;i++)
        free(s[i]);
free(s);
free(q);
free(dataf);
free(tempdataf1);
free(tempdataf2);
free(tempdatay1);
free(tempdatay2);
free(fitness);
for(i=0;i<seeds;i++)
        free(news[i]);
free(news);
free(newq);
free(varflag);
}
```