

1

FILE COPY

FINAL TECHNICAL REPORT
VOLUME II

INTERACTIVE VISUAL SIMULATION OF
COMMUNICATION SYSTEMS

AD-A223 506

DTIC
ELECTE
JUL 06 1990
S D & D

April 29, 1988

Contract No. DAAB07-87-C-A023

Prepared for

Commander

U.S. Army CECOM

Fort Monmouth, N. J. 07703-5000

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

LINKNET

710 Silver Spur Road, Suite 285

Rolling Hills Estates, California 90274

(213) 373-3384

90 07 1 6 7 017

TABLE OF CONTENTS

LINK LIBRARY SOURCE LISTINGS

1.	16QDEM.C	1
2.	16QMOD.C	3
3.	64QDEM.C	4
4.	64QMOD.C	6
5.	8PDEM.C	7
6.	8PMOD.C	9
7.	ADD.C	10
8.	BFDEM.C	11
9.	BFMOD.C	13
10.	BPDEM.C	14
11.	BPMOD.C	15
12.	BSC.C	16
13.	CBSC.C	18
14.	CONV3.C	20
15.	CONV7.C	21
16.	DCONV.C	22
17.	DUNC.C	24
18.	GAUSS.C	26
19.	GEN.C	30
20.	NOCOD.C	31
21.	NODEC.C	32
22.	NO_DEM.C	33
23.	PLOTCONS.C	34
24.	PLOTPB.C	35
25.	PLTRCINP.C	36
26.	PLTRCOUT.C	37
27.	PLTTXINP.C	38
28.	PLTTXOUT.C	39
29.	PROP.C	40
30.	QFDEM.C	46
31.	QFMOD.C	48
32.	QPDEM.C	50
33.	QPMOD.C	51
34.	RCBUT.C	52
35.	RCHEB.C	57
36.	RCELL.C	63
37.	RCFIR.C	70
38.	RSCOD.C	75
39.	RSDEC.C	78
40.	SIN.C	83
41.	SINK.C	85
42.	TBPDEM.C	86
43.	TBPMOD.C	87
44.	TXBUT.C	88
45.	TXCHEB.C	93
46.	TXELL.C	99
47.	TXFIR.C	106
48.	VIT3.C	111
49.	VIT3Q.C	113
50.	VIT7.C	115
51.	VIT7Q.C	117

NETWORK LIBRARY SOURCE LISTINGS

52. ALOHA.PAS		119
53. BIDLOOP.PAS		128
54. CSMA.PAS		141
55. PLTDELAY.C		152
56. PLTTHRPT.C		153
57. STARNET.PAS		154
58. TREE.PAS		167
59. UNILoop.PAS		177

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By <i>per form 50</i>	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
<i>A-1</i>	

```

/*****          16QDEM.C (16QAM Demodulator)          *****/
/* Function qam16_dem() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    FILE *fp;
    int time;
} STATE, *STATEPTR;

qam16_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE Iinput,Qinput,output;
    int i,point;
    float test,min;
    FILE *fopen();

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->time=0;
        pstate->fp=fopen("graf.dat","w");
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 1)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        get(0,&Iinput);
        get(0,&Qinput);

        if(pstate->time<2000) fprintf(pstate->fp,"\n%f, %f\r",Iinput,Qinput);
        if(pstate->time==2000) fclose(pstate->fp);

        min=10000;
        for(i=0;i<4;i++)
        {
            test= (Iinput-(i*2-3))*(Iinput-(i*2-3));
            if(test<min) {min=test; point=i;}
        }
        output=(SAMPLE) (point&01);
        put(0,output);
        output=(SAMPLE) ((point>>1)&01);
        put(0,output);
    }
}

```

```

min=10000;
for(i=0;i<4;i++)
{
test= (Qinput-(i*2-3))*(Qinput-(i*2-3));
if(test<min) {min=test; point=i;}
}
output=(SAMPLE) (point&01);
put(0,output);
output=(SAMPLE) ((point>>1)&01);
put(0,output);

pstate->time++;
}
    return (0) ;
}

```

```

/*****          16QMOD.C (16QAM Modulator)          *****/
/* Function qam16_mod() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define SCALE 1

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

qam16_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,Ioutput,Qoutput;
    int point;

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 3)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        point=0;
        get(0,&input);
        point ^= (input>0);
        get(0,&input);
        point ^= (input>0)*2;
        Ioutput=SCALE*(2*point-3);
        put(0,Ioutput);
        point=0;
        get(0,&input);
        point ^= (input>0);
        get(0,&input);
        point ^= (input>0)*2;
        Qoutput=SCALE*(2*point-3);
        put(0,Qoutput);
    }

    return (0) ;
}

```

```

/*****          64QDEM.C (64QAM Demodulator)          *****/
/* Function qam64_dem() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    FILE *fp;
    int time;
} STATE, *STATEPTR;

qam64_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE Iinput,Qinput,output;
    int i,point;
    float test,min;
    FILE *fopen();

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->time=0;
        pstate->fp=fopen("graf.dat","w");
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 1)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&Iinput);
        get(0,&Qinput);

        if(pstate->time<2000) fprintf(pstate->fp,"\n%f, %f\r",Iinput,Qinput);
        if(pstate->time==2000) fclose(pstate->fp);

        min=10000;
        for(i=0;i<8;i++)
        {
            test= (Iinput-(i*2-7))*(Iinput-(i*2-7));
            if(test<min) {min=test; point=i;}
        }
        for(i=0;i<3;i++) {output=(SAMPLE)((point>>i)&01); put(0,output);}

        min=10000;
        for(i=0;i<8;i++)

```

```

{
test= (Qinput-(i*2-7))*(Qinput-(i*2-7));
if(test<min) {min=test; point=i;}
}
for(i=0;i<3;i++) {output=(SAMPLE)((point>>i)&01); put(0,output);}

pstate->time++;
}
    return (0) ;
}

```

```

/*****          64QMOD.C (64QAM Modulator)          *****/
/* Function gam64_mod() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define SCALE    1

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

gam64_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,Ioutput,Qoutput;
    int point;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }
    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 5)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        point=0;
        get(0,&input);
        point ^= (input>0);
        get(0,&input);
        point ^= (input>0)*2;
        get(0,&input);
        point ^= (input>0)*4;
        Ioutput=SCALE*(2*point-7);
        put(0,Ioutput);
        point=0;
        get(0,&input);
        point ^= (input>0);
        get(0,&input);
        point ^= (input>0)*2;
        get(0,&input);
        point ^= (input>0)*4;
        Qoutput=SCALE*(2*point-7);
        put(0,Qoutput);
    }
    return (0) ;
}

```

```

/*****          8PDEM.C ( 8PSK Demodulator)          *****/
/* Function psk8_dem() */

#include <stdio.h>
#include <mth.h>
#include "type.h"
#include "star.h"
#define THETA 0.785398      /* 2*PI/8 */

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    float x[8],y[8];
    FILE *fp;
    int time;
} STATE, *STATEPTR;

psk8_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE Iinput,Qinput,output;
    int i,point;
    float test,min;
    FILE *fopen();

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        for(i=0;i<8;i++)
            {pstate->x[i]=cos(i*THETA); pstate->y[i]=sin(i*THETA);}
        pstate->fp=fopen("graf.dat","w");
        pstate->time=0;
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 1)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&Iinput);
        get(0,&Qinput);

        if(pstate->time<2000) fprintf(pstate->fp,"\n%f, %f\r",Iinput,Qinput);
        if(pstate->time==2000) fclose(pstate->fp);

        min= -Iinput*pstate->x[0]-Qinput*pstate->y[0];
        point=0;
        for(i=1;i<8;i++)
        {
            test= -Iinput*pstate->x[i]-Qinput*pstate->y[i];

```

```

if(test<min) {min=test; point=i;}
}
output=2*(point&01)-1;
put(0,output);
output=2*((point>>1)&01)-1;
put(0,output);
output=2*((point>>2)&01)-1;
/*
printf("\ndem: %4d  ",point);
printf(" %f %f",Iinput,Qinput);
*/
put(0,output);
pstate->time++;
}
    return (0) ;
}

```

```

/*****          8PMOD.C (8PSK Modulator)          *****/
/* Function psk8_mod() */

#include <stdio.h>
#include <mth.h>
#include "type.h"
#include "star.h"
#define THETA    0.78539816                /*    2*PI/8    */

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

psk8_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,Ioutput,Qoutput;
    int point;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }
    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
    while(length_input_fifo(0) > 2)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        point=0;
        get(0,&input);
        point ^= (input>0);
        get(0,&input);
        point ^= (input>0)*2;
        get(0,&input);
        point ^= (input>0)*4;
        Ioutput=cos(THETA*point);
        Qoutput=sin(THETA*point);
        /*
        printf("8psk_mod: %4d",point);
        printf(" %f %f",Ioutput,Qoutput);
        */
        put(0,Ioutput);
        put(0,Qoutput);
    }
    return (0) ;
}

```

```

/***** ADD.C *****/
/* Function adder() */

#include "type.h"
#include "star.h"
#include <stdio.h>

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

adder(pparam, size, pstate, pstar)
    inc size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE signal, noise, output;
    int none;
    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1, sizeof(STATE));
        if(no_input_fifos() != 2 ) return(1);
        if(no_output_fifos() != 1) return(2);
    }

    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
    if(length_input_fifo(0) <2) return(0);
    if(length_input_fifo(0)%2 != 0) return(56);
    if(length_input_fifo(1) <2) return(0);
    if(length_input_fifo(1)%2 != 0) return(57);
    if(length_input_fifo(0) != length_input_fifo(1)) return(59);

    while(length_input_fifo(1) >1)
    {
        if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
        get(0, &signal);
        get(1, &noise);
        output=signal+noise;
        put(0, output);

        get(0, &signal);
        get(1, &noise);
        output=signal+noise;
        put(0, output);
    }
    return(0);
}

```

```

/***** BFDEM.C *****/
/* Function bfsk_dem() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int time;
    FILE *fp;
} STATE, *STATEPTR;

bfsk_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE Iinp_w1,Iinp_w2,Qinp_w2,Qinp_w1,output;
    float q1,q2;
    FILE *fopen();

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->fp=fopen("graf.dat","w");
        pstate->time=0;
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 3)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&Iinp_w1);
        get(0,&Qinp_w1);
        get(0,&Iinp_w2);
        get(0,&Qinp_w2);
        q1=Iinp_w1*Iinp_w1+Qinp_w1*Qinp_w1;
        q2=Iinp_w2*Iinp_w2+Qinp_w2*Qinp_w2;
        if(q1<q2) output= -1; else output=1;

        if(pstate->time<2000)
        {
            if(q1<q2) fprintf(pstate->fp,"\n%f, %f\r",Iinp_w2,Qinp_w2);
            else fprintf(pstate->fp,"\n%f, %f\r",Iinp_w1,Qinp_w1);
        }
        if(pstate->time==2000) fclose(pstate->fp);
    }
}

```

```
put(0,output);
```

```
pstate->time++;  
}
```

```
    return (0) ;  
}
```

```

/***** BFMOD.C *****/
/* Function bfsk_mod() */

```

```

#include <stdio.h>
#include "type.h"
#include "star.h"
#include <mth.h>
#define PI 3.1415926535

```

```

typedef struct {
    int non;
} PARAM, *PARAMPTR;

```

```

typedef struct {
    double seed;
} STATE, *STATEPTR;

```

```

bfsk_mod (pparam, size, pstate, pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;

```

```

{
    SAMPLE input, Iout_w1, Qout_w1, Iout_w2, Qout_w2;
    double dquo, theta;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if (no_input_fifos( ) != 1 || no_output_fifos( ) != 1) return(3);
        pstate->seed=7;
    }

```

```

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

    while(length_input_fifo(0) > 0)

```

```

    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&input);

```

```

        pstate->seed *= 16807;
        dquo=(int)(pstate->seed/2147483647.0);
        pstate->seed -= dquo*2147483647.0;
        theta=2*PI*pstate->seed/2147483647.0;

```

```

        if(input>0) {Iout_w1=cos(theta); Qout_w1=sin(theta);
                     Iout_w2=0;          Qout_w2=0;}
        else        {Iout_w2=cos(theta); Qout_w2=sin(theta);
                     Iout_w1=0;          Qout_w1=0;}

```

```

        put(0,Iout_w1);
        put(0,Qout_w1);
        put(0,Iout_w2);
        put(0,Qout_w2);

```

```

    }
    return (0) ;
}

```

```
/* ***** BPDEM.C ***** */
/* Function bpsk_dem() */
```

```
#include <stdio.h>
#include "type.h"
#include "star.h"
```

```
typedef struct {
    int non;
} PARAM, *PARAMPTR;
```

```
typedef struct {
    int none;
} STATE, *STATEPTR;
```

```
bpsk_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
```

```
{
    SAMPLE input,output;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }
}
```

```
if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
```

```
while(length_input_fifo(0) > 0)
```

```
{
    if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
    get(0,&input);
```

```
    if(input>0) output=1; else output= -1;
    put(0,output);
```

```
    }
    return (0) ;
}
```

```

/***** BPMOD.C *****/
/* Function bpsk_mod() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

bpsk_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,output;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        get(0,&input);
        output=input;

        put(0,output);
    }
    return (0) ;
}

```

```

/***** BSC.C *****/
/* Function bsc() */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>

typedef struct {
    int samp_stop;
    int seed;
    double stdev;
} PARAM,*PARAMPTR;

typedef struct {
    int sample_no;
    int samp_stop;
    double sample_temp;
    float epsilon;
} STATE,*STATEPTR;

bsc(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    FILE *fopen(),*f1,*f2;
    SAMPLE signal,output;
    float v1;
    double dquo;
    int i;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));
        pstate->sample_no = 0;
        pstate->sample_temp=pparam->seed;
        if(size == 0)                pstate->samp_stop = 100;
        else if(size == sizeof(PARAM)) pstate->samp_stop = pparam->samp_stop;
        else
            return(1);

        f1=fopen("channel.tdt","r");
        f2=fopen("pbresult.dat","w");
        fscanf(f1, "%f",&pstate->epsilon);
        fclose(f1);

        printf("\n BINARY SYMMETRIC CHANNEL \n");
        fprintf(f2,"\n BINARY SYMMETRIC CHANNEL \n \r");
        printf("\n BSC Crossover Probability = %9.6f \n",pstate->epsilon);
        fprintf(f2,"\n BSC Crossover Probability = %9.6f \n \r",pstate->epsilon);
        fclose(f2);

        if(no_output_fifos() != 1)    return(2);
        if(no_input_fifos() != 1)    return(3);
    }
}

```

```

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(pstate->sample_no >= pstate->samp_stop)    return(99);
    while(length_input_fifo(0) >0)
    {
        if(length_output_fifo(0) == maxlength_output_fifo(0))    return(0);
get(0,&signal);
        pstate->sample_temp *= 16807;
        dquo=(int)(pstate->sample_temp/2147483647.0);
        pstate->sample_temp -= dquo*2147483647.0;
        v1=pstate->sample_temp/2147483647.0;
        if(v1>pstate->epsilon)    output=  signal;
        else    output= -signal;
put(0,output);
        pstate->sample_no++;
    }
return(0);
}

```

```

/***** CBSC.C *****/
/* Function bsc() */

```

```

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>

```

```

typedef struct {
    int samp_stop;
    int seed;
    double stdev;
} PARAM, *PARAMPTR;

```

```

typedef struct {
    int sample_no;
    int samp_stop;
    double sample_temp;
    float epsilon;
} STATE, *STATEPTR;

```

```

bsc(pparam, size, pstate, pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;

```

```

{
    FILE *fopen(), *f1, *f2;
    SAMPLE signal, output;
    float v1;
    double dquo;
    int i;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1, sizeof(STATE));
        pstate->sample_no = 0;
        pstate->sample_temp=pparam->seed;
        if(size == 0)                pstate->samp_stop = 100;
        else if(size == sizeof(PARAM)) pstate->samp_stop = pparam->samp_stop;
        else
            return(1);

        f1=fopen("channel.tdt", "r");
        f2=fopen("pbresult.dat", "w");
        fscanf(f1, "%f", &pstate->epsilon);
        fclose(f1);

        printf("\n BINARY SYMMETRIC CHANNEL \n");
        fprintf(f2, "\n BINARY SYMMETRIC CHANNEL \n \r");
        printf("\n BSC Crossover Probability = %9.6f \n", pstate->epsilon);
        fprintf(f2, "\n BSC Crossover Probability = %9.6f \n \r", pstate->epsilon);
        fclose(f2);

        if(no_output_fifos() != 1)    return(2);
        if(no_input_fifos() != 1)    return(3);
    }
}

```

```

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(pstate->sample_no >= pstate->samp_stop)    return(99);

    while(length_input_fifo(0) >1)
    {
        if(length_output_fifo(0) == maxlength_output_fifo(0))    return(0);

get(0,&signal);
    pstate->sample_temp *= 16807;
    dquo=(int)(pstate->sample_temp/2147483647.0);
    pstate->sample_temp -= dquo*2147483647.0;
    v1=pstate->sample_temp/2147483647.0;

    if(v1>pstate->epsilon)    output=  signal;
    else    output= -signal;

    put(0,output);

get(0,&signal);
    pstate->sample_temp *= 16807;
    dquo=(int)(pstate->sample_temp/2147483647.0);
    pstate->sample_temp -= dquo*2147483647.0;
    v1=pstate->sample_temp/2147483647.0;

    if(v1>pstate->epsilon)    output=  signal;
    else    output= -signal;

    put(0,output);

        pstate->sample_no++;
    }
return(0);
}

```

```

/***** CONV3.C (ENCODER FOR (3,1/2) CONVOLUTIONAL CODE) *****/
/* Function conv() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int s[3];
} STATE, *STATEPTR;

conv (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,output0,output1;
    int i;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=2) return(3);
        for(i=0;i<3;i++) pstate->s[i]=0;
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
    if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);
        get(0,&input);
        put(1,input);

        for(i=1;i>=0;i--) pstate->s[i+1]=pstate->s[i];
        pstate->s[0]=(int)input;
        output0=(pstate->s[0]+pstate->s[2])%2;
        output1=(pstate->s[0]+pstate->s[1]+pstate->s[2])%2;
        output0=2*output0-1;
        output1=2*output1-1;

        put(0,output0);
        put(0,output1);
    }
    return (0) ;
}

```

```

/***** CONV7.C (ENCODER FOR (7,1/2) CONVOLUTIONAL CODE) *****/
/* Function conv() */

```

```

#include <stdio.h>
#include "type.h"
#include "star.h"

```

```

typedef struct {
    int non;
} PARAM, *PARAMPTR;

```

```

typedef struct {
    int s[7];
} STATE, *STATEPTR;

```

```

conv (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;

```

```

{
    SAMPLE input,output0,output1;
    int i;

```

```

        if (pstate == NULL) {
            pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
            if (no_input_fifos( ) !=1 || no_output_fifos( ) !=2) return(3);
            for(i=0;i<7;i++) pstate->s[i]=0;
        }

```

```

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
    if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);

```

```

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        if (length_output_fifo(1)==maxlength_output_fifo(1))    return(0);
        get(0,&input);
        put(1,input);

```

```

        for(i=5;i>=0;i--) pstate->s[i+1]=pstate->s[i];
        pstate->s[0]=(int)input;
        output0=(pstate->s[0]+pstate->s[2]+pstate->s[5]+pstate->s[6]+pstate->s[3])%2;
        output1=(pstate->s[0]+pstate->s[1]+pstate->s[2]+pstate->s[3]+pstate->s[6])%2;

```

```

        output0=2*output0-1;
        output1=2*output1-1;
        put(0,output0);
        put(0,output1);
    }

```

```

        return (0) ;
    }

```

```

/***** DCONV.C *****/
/* Function display() */
/* Calculates error probability for Viterbi decoded convolutional
   coded systems. Writes results to PBRESULT.DAT file.
*/

#include "type.h"
#include "star.h"
#include <stdio.h>
#define LT 32

typedef struct {
    int sample_stp;
    int seed;
} PARAM, *PARAMPTR;

typedef struct {
    float time; /* holds current time, which is printed */
    float time_scale;
    int errdc, errva;
    int vi[LT];
    int log2M;
    int ser, ser2;
    int sample_stp;
    FILE *fo, *f1, *f2;
} STATE, *STATEPTR;

display(pparam, size, pstate, pstar)

PARAMPTR pparam;
int size; /* size of parameter storage */
STATEPTR pstate;
STARPTR pstar;

{
    SAMPLE x, y;
    FILE *fopen(), *fp;
    int i, j;
    float disp, tmp;

    if(pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if(no_input_fifos() != 2) return(1);
        if(no_output_fifos() != 0) return(2);
        pstate->time_scale = 1.0;
    }

    if(size <= 0) return(204);
    pstate->sample_stp = pparam->sample_stp;
    pstate->time = 0.0;
    pstate->ser = 0;
    pstate->ser2 = 0;
    pstate->errva = 0;
    pstate->errdc = 0;
    fp = fopen("modsize.tdt", "r");

```

```

        pstate->fo=fopen("pbresult.dat","a");
        pstate->f1=fopen("pb.dat","w");
        pstate->f2=fopen("ps.dat","w");
        fscanf(fp,"%f",&tmp);
        pstate->log2M=(int)tmp;
        fclose(fp);

        /*printf("\nComparison Delay = %d\n\n",LT);*/
        printf("\n \n");
        for(i=0;i<LT;i++) pstate->vi[i]=0;
    }

    if(length_input_fifo(0) != length_input_fifo(1)) return(0);
    while(length_input_fifo(0)>0)
    {
        get(1,&x);
        get(0,&y);
        if(((int)pstate->time)%pstate->log2M==0)
        {if(pstate->ser>0) pstate->ser2++; pstate->ser=0;}
        for(i=LT-2;i>=0;i--) pstate->vi[i+1]=pstate->vi[i];
        pstate->vi[0]=(int)x;
        if(pstate->vi[LT-1] != (int)y)
        {pstate->ser++; pstate->errva++; }

        if((int)pstate->time>0)
            if((int)pstate->time%(pstate->sample_stp/100)==0) {
                disp=pstate->errva/pstate->time;
                printf("Pb=%9.6f ",disp);
                fprintf(pstate->f1,"\n %9.6f, %9.6f\r",pstate->time,disp);
                if((int)pstate->time == pstate->sample_stp) {
                    fprintf(pstate->fo,"\n \r");
                    fprintf(pstate->fo,"\n Bit Error Probability = %9.6f \r",disp);
                }

                disp=pstate->ser2/(pstate->time/pstate->log2M);
                printf(" Ps=%9.6f \r",disp);
                fprintf(pstate->f2,"\n %9.6f, %9.6f\r",pstate->time,disp);
                if((int)pstate->time == pstate->sample_stp) {
                    fprintf(pstate->fo,"\n \r");
                    fprintf(pstate->fo,"\n Symbol Error Probability = %9.6f \r",disp);
                }
            }

        pstate->time += 1;
    }

    return(0);
}

```

```

/***** DUNC.C *****/
/* Function display() */
/* Calculates error probability for uncoded and Reed Solomon
   coded systems. Writes results to PBRESULT.DAT file.
*/

#include "type.h"
#include "star.h"
#include <stdio.h>

typedef struct {
    int sample_stp;
    int seed;
} PARAM, *PARAMPTR;

typedef struct {
    float time; /* holds current time, which is printed */
    float time_scale;
    int errva;
    int log2M;
    int ser,ser2;
    int sample_stp;
    FILE *fo, *f1, *f2;
} STATE, *STATEPTR;

display(pparam,size,pstate,pstar)

PARAMPTR pparam;
int size; /* size of parameter storage */
STATEPTR pstate;
STARPTR pstar;

{
    SAMPLE x,y;
    FILE *fopen(),*fp;
    int i,j;
    float disp,tmp;

    if(pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if(no_input_fifos() != 2) return(1);
        if(no_output_fifos() != 0) return(2);
        pstate->time_scale = 1.0;
    }

    if(size<=0) return(204);

    pstate->sample_stp=pparam->sample_stp;
    pstate->time = 0.0;
    pstate->ser=0;
    pstate->ser2=0;
    pstate->errva=0;
    fp=fopen("modsize.tdt","r");
    pstate->fo=fopen("pbresult.dat","a");
    pstate->f1=fopen("pb.dat","w");
    pstate->f2=fopen("ps.dat","w");

```

```

        fscanf(fp,"%f",&tmp);
        pstate->log2M=(int)tmp;
        fclose(fp);

        printf("\n \n");
    }

    if(length_input_fifo(0) != length_input_fifo(1)) return(58);
    while(length_input_fifo(0)>0)
    {
        get(1,&x);
        get(0,&y);
        if(((int)pstate->time)%pstate->log2M==0)
        {if(pstate->ser>0) pstate->ser2++; pstate->ser=0;}
        if((int)x != (int)y)
        {pstate->ser++; pstate->errva++; }

        if((int)pstate->time>0)
            if((int)pstate->time%(pstate->sample_stp/100)==0) {
                disp=pstate->errva/pstate->time;
                printf("Pb=%9.6f ",disp);
                fprintf(pstate->f1,"\n %9.6f, %9.6f\r",pstate->time,disp);
                if((int)pstate->time == pstate->sample_stp) {
                    fprintf(pstate->fo,"\n \r");
                    fprintf(pstate->fo,"\n Bit Error Probability = %9.6f \r",disp);
                }

                disp=pstate->ser2/(pstate->time/pstate->log2M);
                printf(" Ps=%9.6f \r",disp);
                fprintf(pstate->f2,"\n %9.6f, %9.6f\r",pstate->time,disp);
                if((int)pstate->time == pstate->sample_stp) {
                    fprintf(pstate->fo,"\n \r");
                    fprintf(pstate->fo,"\n Symbol Error Probability = %9.6f \r",disp);
                }
            }
    }

    pstate->time += 1;
}

return(0);
}

```

```

/***** GAUSS.C *****/
/* Function gauss() */
/* This function generates a sequence of Gaussian random variables
of given Std. dev. and zero mean. File Parameter Version.
*/

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>

typedef struct {
    int samp_stop, seed;
    double stddev;
} PARAM, *PARAMPTR;

typedef struct {
    int sample_no; /* current sample number */
    int samp_stop; /* stop after this no. of samples */
    double sample_temp;
} STATE, *STATEPTR;

gauss(pparam, size, pstate, pstar)

PARAMPTR pparam;
int size; /* size of parameter storage */
STATEPTR pstate;
STARPTR pstar;
{
    float v1, v2, t, SNo, Rb, snr, EbNo, code_rate, tmp, fc;
    int log2M, L, L1, L2;
    double dquo;
    FILE *fopen(), *f1, *f2, *f3, *f4, *f5, *f6, *f7, *f8;
    SAMPLE bis;

    if(pstate == NULL)
    {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        pstate->sample_no = 0;
        pstate->sample_temp = pparam->seed;
        if(size == 0) pstate->samp_stop = 100;
        else if(size == sizeof(PARAM)) pstate->samp_stop = pparam->samp_stop;
        else return(1);

        /* First determine whether source is random bit */

        f5 = fopen("sou_con.tdt", "r");
        fscanf(f5, "%f", &tmp);
        L2 = (int)tmp;
        fclose(f5);
        /* L2 = 0 for random bit, L2=1 otherwise */

        /*printf("\n L2 = %d ", L2);*/

        /* Next read in S/No */
    }
}

```

```

f2=fopen("channel.tdt","r");    /* SNo */
fscanf(f2,"%f",&SNo);
fclose(f2);

/* Open Results Output File PBRESULT.DAT */

f8=fopen("pbresult.dat","w");

printf("\n ADDITIVE WHITE GAUSSIAN NOISE CHANNEL");
printf("\n S/No = %10.2f dB ",SNo);

fprintf(f8,"\n ADDITIVE WHITE GAUSSIAN NOISE CHANNEL \r");
fprintf(f8,"\n \r");
fprintf(f8,"\n S/No = %10.2f dB \r",SNo);

if(L2 == 0) {
    /*printf("\n Random Bit Source");*/
    f1=fopen("source.tdt","r");    /* Rb */
    f3=fopen("coderate.tdt","r");    /* code_rate */
    f4=fopen("modsize.tdt","r");    /* log2M */

    fscanf(f1,"%f",&Rb);
    fscanf(f3,"%f",&code_rate);
    fscanf(f4,"%f",&tmp);
    log2M=(int)tmp;
    fscanf(f4,"%f",&tmp);
    L=(int)tmp;

    /* L = 0 for signal vector modulations, L = 1 for time domain
    modulations
    */

    fclose(f1);
    fclose(f3);
    fclose(f4);

    printf("\n Rb = %10.2f bps ",Rb);
    printf("\n code_rate = %10.2f ",code_rate);
    /* printf("\n log2M = %d ",log2M); */

    fprintf(f8,"\n Rb = %10.2f bps \r",Rb);
    fprintf(f8,"\n code_rate = %10.2f \r",code_rate);

    if(L == 0) {
        /*printf("\n Signal Vector Modulation");*/
        snr=SNo-10*log10(Rb/(code_rate*log2M));
        EbNo=snr-10*log10(code_rate*log2M);

        printf("\n Es/No = %10.2f dB ",snr);
        printf("\n Eb/No = %10.2f dB ",EbNo);

        fprintf(f8,"\n Es/No = %10.2f dB \r",snr);
        fprintf(f8,"\n Eb/No = %10.2f dB \r",EbNo);
    }
}

```

```

pparam->stddev=sqrt(0.5*exp(-ln(10.0)*snr/10.0));

if(log2M == 4)
    pparam->stddev *= sqrt(10.0);

if(log2M == 6)
    pparam->stddev *= sqrt(42.0);

printf("\n Standard deviation of noise= %f\n",pparam->stddev);
fprintf(f8,"\n Standard deviation of noise= %f \r \n \r",pparam->stddev);

    } /* end if(L == 0) */

if(L == 1) {
    /*printf("\n Filtered Time Domain Modulations");*/
    f6=fopen("ch1fltr1.tdt","r");
    f7=fopen("ch1fltr2.tdt","r");
    fscanf(f6,"%f",&tmp);
    L1=(int)tmp;

    if(L1 == 1) {
        /*printf("\n Filter Spec by Order");*/
        fscanf(f6,"%f",&tmp);
        fscanf(f6,"%f",&tmp);
        fscanf(f6,"%f",&fc);
    } /* end if (L1 == 1) */

    else {
        /*printf("\n Filter Spec by Attenuation");*/
        fscanf(f7,"%f",&tmp);
        fscanf(f7,"%f",&fc);
    } /* end else */

    fclose(f6);
    fclose(f7);

    snr=SNo - 10*log10(fc);
    printf("\n S/N = %10.2f dB ",snr);
    fprintf(f8,"\n S/N = %10.2f dB \r",snr);
    pparam->stddev=sqrt(exp(-ln(10.0)*snr/10.0));

    printf("\n Standard deviation of noise= %f\n",pparam->stddev);
    fprintf(f8,"\n Standard deviation of noise= %f \r \n \r",pparam->stddev);

        } /* end if(L == 1) */

    fclose(f8);

    } /* end if(L2 == 0) */

if(L2 == 1) {
    /*printf("\n Sinuisoidal Source");*/

```

```

        fl=fopen("source.tdt","r");
        fscanf(fl,"%f",&fc);          /* read in sinuisoidal fs */
        fclose(fl);

        snr=SNo - 10*log10(fc);
        printf("\n S/N = %10.2f  dB ",snr);
        pparam->stddev=sqrt(exp(-ln(10.0)*snr/10.0));

    printf("\n Standard deviation of noise= %f\n",pparam->stddev);

        } /* end if(L2 == 1) */

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 0)    return(3);

    } /* end if(pstate == NULL) */

    if(pstate->sample_no >= pparam->samp_stop)    return(99);
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

do {
    do {

        pstate->sample_temp *= 16807;
        dquo=(int)(pstate->sample_temp/2147483647.0);
        pstate->sample_temp -= dquo*2147483647.0;
        v1=2*(pstate->sample_temp/2147483647.0)-1;
        pstate->sample_temp *= 16807;
        dquo=(int)(pstate->sample_temp/2147483647.0);
        pstate->sample_temp -= dquo*2147483647.0;
        v2=2*(pstate->sample_temp/2147483647.0)-1;
        t=v1*v1+v2*v2;
    }
        while(t >= 1);

    bis=(SAMPLE)((pparam->stddev)*v1*sqrt((-2)*ln(t)/t));
    (pstate->sample_no)++;
}
while(put(0,bis) ==0) ;

        return(0);          /* normal return */
    }

```

```

/***** GEN.C *****/
/* Function gen() */
/* Generates a sequence of random bits (0,1) */

#include "type.h"
#include "star.h"
#include <stdio.h>

typedef struct {
    int sample_stp;
    int seed;
} PARAM, *PARAMPTR;

typedef struct {
    int sample_no;
    int sample_stop;
    double sample_temp;
} STATE, *STATEPTR;

gen(pparam,size,pstate,pstar)
PARAMPTR pparam;
int size;
STATEPTR pstate;
STARPTR pstar;
{
    double dquo,u;
    SAMPLE bit;
    if(pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        pstate->sample_no = 0;
        pstate->sample_temp=pparam->seed;
        if(size == 0) pstate->sample_stop = 100;
        else if(size == sizeof(PARAM))
            pstate->sample_stop = pparam->sample_stp;
        else return(1);
        if(no_output_fifos() != 1) return(2);
        if(no_input_fifos() != 0) return(3);
    }
    if(pstate->sample_no > pparam->sample_stp) return(99);
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

    while(length_output_fifo(0)<maxlength_output_fifo(0))
    {
        pstate->sample_temp *= 16807;
        dquo=(int) (pstate->sample_temp/2147483647.0);
        pstate->sample_temp -= dquo*2147483647.0;
        u=pstate->sample_temp/2147483647.0;
        bit=(u>=0.5);
        put(0,bit);
        pstate->sample_no++;
    }
    return(0);
}

```

```
/* ***** NOCOD.C ***** */
```

```
/* Function nocod() */
```

```
#include <stdio.h>
```

```
#include "type.h"
```

```
#include "star.h"
```

```
typedef struct {  
    int non;  
} PARAM, *PARAMPTR;
```

```
typedef struct {  
    int none;  
} STATE, *STATEPTR;
```

```
nocod (pparam,size,pstate,pstar)
```

```
    PARAMPTR pparam;
```

```
    STATEPTR pstate;
```

```
    STARPTR pstar;
```

```
    int size;
```

```
{  
    SAMPLE input,output;
```

```
    if (pstate == NULL) {
```

```
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
```

```
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=2) return(3);
```

```
    }
```

```
    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
```

```
    if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);
```

```
    while(length_input_fifo(0) > 0)
```

```
    {  
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
```

```
        if (length_output_fifo(1)==maxlength_output_fifo(1))    return(0);
```

```
        get(0,&input);
```

```
        put(1,input);
```

```
        output=2*input-1;
```

```
        put(0,output);
```

```
    }
```

```
        return (0) ;
```

```
    }
```

```

/***** NODEC.C *****/
/* Function nodec() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

nodec (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,output;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        get(0,&input);
        output=((int)input)+1)/2;
        put(0,output);
    }

    return (0) ;
}

```

```

/***** NO_DEM.C *****/
/* Function no_dem() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int time;
    FILE *fp;
} STATE, *STATEPTR;

no_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,output;
    FILE *fopen();

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->time=0;
        pstate->fp=fopen("graf.dat","w");
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0))    return(0);
        get(0,&input);

        if(pstate->time<2000)
        {
            if(pstate->time%2==0) fprintf(pstate->fp,"\n%f",input);
            else
                fprintf(pstate->fp," %f\r",input);
        }
        if(pstate->time==2000) fclose(pstate->fp);

        output=input;
        put(0,output);
        pstate->time++;
    }

    return (0) ;
}

```

```
/****** PLOTCONS.C *****/
```

```
#include <stdio.h>
```

```
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
```

```
{  
    int count, dev;  
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info */  
    auto_select_display();  
    virtual_display(YES);  
    /* graph data in "graf.dat" */  
    get_pair_info("graf.dat", &count, &xmin, &xmax, &ymin, &ymax);  
    hrange((xmin - xmax)/2 - 1.0, (xmax - xmin)/2 + 1.0);  
    vrange((ymin - ymax)/2 - 1.0, (ymax - ymin)/2 + 1.0);  
    graph_type(ORDINARY);  
    line_connect(NO);  
    display_onscreen(YES);  
    title("SIGNAL VECTOR CONSTELLATION", CENTER);  
    haxis_lbl("I AXIS", CENTER);  
    vaxis_lbl("Q AXIS", CENTER);  
    display_window(0,900,0,643);  
    graph_init();  
    plot_pairs(1, "graf.dat");  
    graph_close();  
}
```

```
/****** PLOTPB.C *****/
```

```
#include <stdio.h>
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
```

```
{
    int count, dev;
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info */
    auto_select_display();
    virtual_display(YES);
    /* graph data in "pb.dat" */
    get_pair_info("pb.dat", &count, &xmin, &xmax, &ymin, &ymax);
    hrange(0.0, xmax);
    vrange(ymin, ymax);
    graph_type(ORDINARY);
    line_connect(YES);
    display_onscreen(YES);
    hlbl_prec(0);
    vlbl_prec(6);
    title("Bit Error Probability", CENTER);
    haxis_lbl("Simulation Time", CENTER);
    vaxis_lbl("Bit Error Probability", CENTER);
    display_window(0,900,0,643);
    graph_init();
    plot_pairs(1, "pb.dat");
    graph_close();
}
```

```
/****** PLTRCINP.C *****/
```

```
#include <stdio.h>
```

```
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
```

```
{
```

```
    int count, dev;
```

```
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info */
```

```
    auto_select_display();
```

```
    virtual_display(YES);
```

```
    get_pair_info("rcinp.dat", &count, &xmin, &xmax, &ymin, &ymax);
```

```
    hrange(0.0, xmax);
```

```
    vrange((ymin - ymax)/2 - 0.25, (ymax - ymin)/2 + 0.25);
```

```
    graph_type(ORDINARY);
```

```
    line_connect(YES);
```

```
    display_onscreen(YES);
```

```
    hlbl_prec(0);
```

```
    vlbl_prec(2);
```

```
    title("Signal Waveform", CENTER);
```

```
    haxis_lbl("Time", CENTER);
```

```
    vaxis_lbl("Amplitude", CENTER);
```

```
    display_window(0,900,0,643);
```

```
    graph_init();
```

```
    plot_pairs(1, "rcinp.dat");
```

```
    graph_close();
```

```
}
```

```
/****** PLTRCOUT.C *****/
```

```
#include <stdio.h>
```

```
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
```

```
{
```

```
    int count, dev;
```

```
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info */
```

```
    auto_select_display();
```

```
    virtual_display(YES);
```

```
    get_pair_info("rcout.dat", &count, &xmin, &xmax, &ymin, &ymax);
```

```
    hrange(0.0, xmax);
```

```
    vrange((ymin - ymax)/2 - 0.25, (ymax - ymin)/2 + 0.25);
```

```
    graph_type(ORDINARY);
```

```
    line_connect(YES);
```

```
    display_onscreen(YES);
```

```
    hlbl_prec(0);
```

```
    vlbl_prec(2);
```

```
    title("Signal Waveform", CENTER);
```

```
    haxis_lbl("Time", CENTER);
```

```
    vaxis_lbl("Amplitude", CENTER);
```

```
    display_window(0,900,0,643);
```

```
    graph_init();
```

```
    plot_pairs(1, "rcout.dat");
```

```
    graph_close();
```

```
}
```

```
/****** PLTTXINP.C *****/
```

```
#include <stdio.h>
```

```
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
```

```
{
```

```
    int count, dev;
```

```
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info */
```

```
    auto_select_display();
```

```
    virtual_display(YES);
```

```
    get_pair_info("txinp.dat", &count, &xmin, &xmax, &ymin, &ymax);
```

```
    hrange(0.0, xmax);
```

```
    vrange((ymin - ymax)/2 - 0.25, (ymax - ymin)/2 + 0.25);
```

```
    graph_type(ORDINARY);
```

```
    line_connect(YES);
```

```
    display_onscreen(YES);
```

```
    hlbl_prec(0);
```

```
    vlbl_prec(2);
```

```
    title("Signal Waveform", CENTER);
```

```
    haxis_lbl("Time", CENTER);
```

```
    vaxis_lbl("Amplitude", CENTER);
```

```
    display_window(0,900,0,643);
```

```
    graph_init();
```

```
    plot_pairs(1, "txinp.dat");
```

```
    graph_close();
```

```
}
```

```
/****** PLTTXOUT.C *****/
```

```
#include <stdio.h>
```

```
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
```

```
{  
    int count, dev;  
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info */  
    auto_select_display();  
    virtual_display(YES);  
    get_pair_info("txout.dat", &count, &xmin, &xmax, &ymin, &ymax);  
    hrange(0.0, xmax);  
    vrange((ymin - ymax)/2 - 0.25, (ymax - ymin)/2 + 0.25);  
    graph_type(ORDINARY);  
    line_connect(YES);  
    display_onscreen(YES);  
    hlbl_prec(0);  
    vlbl_prec(2);  
    title("Signal Waveform", CENTER);  
    haxis_lbl("Time", CENTER);  
    vaxis_lbl("Amplitude", CENTER);  
    display_window(0,900,0,643);  
    graph_init();  
    plot_pairs(1, "txout.dat");  
    graph_close();  
}
```

```

/***** PROP.C *****/
/* Function gauss() */
/* File parameter version with Receiver Noise Power Parameter */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>

typedef struct {
    int samp_stop, seed;
    double stddev;
} PARAM, *PARAMPTR;

typedef struct {
    int sample_no; /* current sample number */
    int samp_stop; /* stop after this no. of samples */
    double sample_temp;
} STATE, *STATEPTR;

gauss(pparam, size, pstate, pstar)

PARAMPTR pparam;
int size; /* size of parameter storage */
STATEPTR pstate;
STARPTR pstar;
{
    float v1, v2, t;
    float tmp;
    float K1, K2, Rb, snr, EbNo, code_rate, SNo, No, Cr, a1, a2, a3, a4, a5;
    float HT, HR, ST, GT, GR, fc, r, N, B;
    int AT, FC, AT_FC, log2M, L, L1, L2;
    double dquo;
    FILE *fopen(), *f1, *f2, *f3, *f4, *f5, *f6, *f7, *f8;
    SAMPLE bis;

    if(pstate == NULL)
    {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        pstate->sample_no = 0;
        pstate->sample_temp = pparam->seed;
        if(size == 0) pstate->samp_stop = 100;
        else if(size == sizeof(PARAM)) pstate->samp_stop = pparam->samp_stop;
        else return(1);

        /* First determine whether source is random bit */

        f5 = fopen("sou_con.tdt", "r");
        fscanf(f5, "%f", &tmp);
        L2 = (int)tmp;
        fclose(f5);
        /* L2 = 0 for random bit, L2=1 otherwise */

        /*printf("\n L2 = %d ", L2);*/
    }
}

```

```

/* Next Read in Channel Propagation Parameters to Calculate S/No */
f2=fopen("channel.tdt","r"); /* Propagation Parameters */

/* Open Results Output File PBRESULT.DAT */
f8=fopen("pbresult.dat","w");

printf("\n PROPAGATION CHANNEL");
fprintf(f8,"\n PROPAGATION CHANNEL \r");
fprintf(f8,"\n \r");

/* Reading the Channel Propagation Parameters from File CHANNEL.TDT */

fscanf(f2,"%f",&tmp);
AT_FC=(int)tmp;

if(AT_FC == 0){
    AT = 1;
    FC = 1;}
else if(AT_FC == 1){
    AT = 2;
    FC = 1;}
else if(AT_FC == 2){
    AT = 1;
    FC = 2;}
else {
    AT = 2;
    FC = 2;}

fscanf(f2,"%f",&HT);
fscanf(f2,"%f",&HR);
fscanf(f2,"%f",&ST);
fscanf(f2,"%f",&GT);
fscanf(f2,"%f",&GR);
fscanf(f2,"%f",&fc);
fscanf(f2,"%f",&r);
fscanf(f2,"%f",&N);
fscanf(f2,"%f",&B);
fclose(f2);

/* Print out all Propagation Parameters */

printf("\n USER CHANNEL SPECIFICATIONS");
printf("\n Transmitter Antenna Height = %10.2f meters ",HT);
printf("\n Receiver Antenna Height = %10.2f meters ",HR);
printf("\n Transmitter Average Power = %10.2f watts ",ST);
printf("\n Transmitter Antenna Gain, relative to 1/2 lambda dipole = %10.2f dB",GT);
printf("\n Receiver Antenna Gain, relative to 1/2 lambda dipole = %10.2f dB",GR);
printf("\n Carrier Frequency = %10.2f MHz ",fc);
printf("\n Distance between Transmitter and Receiver = %10.2f Km. ",r);
printf("\n Receiver Noise Power = %10.2f dBm ",N);
printf("\n One-Sided Receiver Noise Bandwidth = %10.2f Hz ",B);

```

```

if(AT==1) printf("\n Open Area"); else printf("\n Suburban Area");
if(FC==1) printf("\n No foliage"); else printf("\n Foliaged");
printf("\n");

fprintf(f8,"\n USER CHANNEL SPECIFICATIONS \r");
fprintf(f8,"\n \r");
fprintf(f8,"\n Transmitter Antenna Height = %10.2f meters \r",HT);
fprintf(f8,"\n Receiver Antenna Height = %10.2f meters \r",HR);
fprintf(f8,"\n Transmitter Average Power = %10.2f watts \r",ST);
fprintf(f8,"\n Transmitter Antenna Gain, relative to 1/2 lambda dipole = ");
fprintf(f8,"%10.2f dB \r",GT);
fprintf(f8,"\n Receiver Antenna Gain, relative to 1/2 lambda dipole = %10.2f
dB \r",GR);
fprintf(f8,"\n Carrier Frequency = %10.2f MHz \r",fc);
fprintf(f8,"\n Distance between Transmitter and Receiver = %10.2f Km. \r",r);
fprintf(f8,"\n Receiver Noise Power = %10.2f dBm \r",N);
fprintf(f8,"\n One-Sided Receiver Noise Bandwidth = %10.2f Hz \r",B);
if(AT==1) fprintf(f8,"\n Open Area \r"); else fprintf(f8,"\n Suburban Area
\r");
if(FC==1) fprintf(f8,"\n No foliage \r"); else fprintf(f8,"\n Foliaged \r");
fprintf(f8,"\n \r");

/* Calculate S/No from the Propagation Parameters */

No=exp(((N/10.0)-3)*ln(10.0))/B;

if(AT==1)
    K1= -79-43.5*log10(r/1.609)-30*log10(fc/900.0);
else
    K1= -91.7-38.4*log10(r/1.609)-30*log10(fc/900.0);

if(FC==1)
    K2= K1;
else
    K2=K1-10;

a1=HT/30.48;
a1 *= a1;
a2=HR/3;
a2 *= a2;
a3=ST/10;
a4=GT-10*log10(4.0);

if(AT==1)
    a5=GR-3;
else
    a5=GR-2;

Cr=K2+10*log10(a1*a2*a3)+a4+a5;
SNo=Cr-10*log10(No);

printf("\n CHANNEL PARAMETERS");
printf("\n S/No = %10.2f dB ",SNo);

```

```

fprintf(f8, "\n CHANNEL PARAMETERS \r");
fprintf(f8, "\n \r");
fprintf(f8, "\n S/No = %10.2f dB \r", SNo);

if(L2 == 0) {
    /*printf("\n Random Bit Source");*/
    f1=fopen("source.tdt", "r");    /* Rb */
    f3=fopen("coderate.tdt", "r");  /* code_rate */
    f4=fopen("modsize.tdt", "r");   /* log2M */

    fscanf(f1, "%f", &Rb);
    fscanf(f3, "%f", &code_rate);
    fscanf(f4, "%f", &tmp);
    log2M=(int)tmp;
    fscanf(f4, "%f", &tmp);
    L=(int)tmp;

    /* L = 0 for signal vector modulations, L = 1 for time domain
       modulations
    */

    fclose(f1);
    fclose(f3);
    fclose(f4);

    printf("\n Rb = %10.2f   bps ", Rb);
    printf("\n code_rate = %10.2f   ", code_rate);
    /*printf("\n log2M = %d   ", log2M);*/

    fprintf(f8, "\n Rb = %10.2f bps \r", Rb);
    fprintf(f8, "\n code_rate = %10.2f \r", code_rate);

    if(L == 0) {
        /*printf("\n Signal Vector Modulation");*/
        snr=SNo-10*log10(Rb/(code_rate*log2M));
        EbNo=snr-10*log10(code_rate*log2M);

        printf("\n Es/No = %10.2f   dB ", snr);
        printf("\n Eb/No = %10.2f   dB ", EbNo);

        fprintf(f8, "\n Es/No = %10.2f dB \r", snr);
        fprintf(f8, "\n Eb/No = %10.2f dB \r", EbNo);

        pparam->stddev=sqrt(0.5*exp(-ln(10.0)*snr/10.0));

        if(log2M == 4)
            pparam->stddev *= sqrt(10.0);

        if(log2M == 6)
            pparam->stddev *= sqrt(42.0);
    }
}

```

```

printf("\n Standard deviation of noise= %f\n",pparam->stddev);
fprintf(f8,"\n Standard deviation of noise= %f \r \n \r",pparam->stddev);

    } /* end if(L == 0) */

    if(L == 1) {
        /*printf("\n Filtered Time Domain Modulations");*/
        f6=fopen("ch1fltr1.tdt","r");
        f7=fopen("ch1fltr2.tdt","r");
        fscanf(f6,"%f",&tmp);
        L1=(int)tmp;

        if(L1 == 1) {
            /*printf("\n Filter Spec by Order");*/
            fscanf(f6,"%f",&tmp);
            fscanf(f6,"%f",&tmp);
            fscanf(f6,"%f",&fc);
        } /* end if (L1 == 1) */

        else {
            /*printf("\n Filter Spec by Attenuation");*/
            fscanf(f7,"%f",&tmp);
            fscanf(f7,"%f",&fc);
        } /* end else */

        fclose(f6);
        fclose(f7);

        snr=SNo - 10*log10(fc);
        printf("\n S/N = %10.2f   dB ",snr);
        fprintf(f8,"\n S/N = %10.2f dB \r",snr);
        pparam->stddev=sqrt(exp(-ln(10.0)*snr/10.0));

    printf("\n Standard deviation of noise= %f\n",pparam->stddev);
    fprintf(f8,"\n Standard deviation of noise= %f \r \n \r",pparam->stddev);

        } /* end if(L == 1) */
    fclose(f8);

    } /* end if(L2 == 0) */

    if(L2 == 1) {
        /*printf("\n Sinuisoidal Source");*/
        f1=fopen("source.tdt","r");
        fscanf(f1,"%f",&fc);          /* read in sinuisoidal fs */
        fclose(f1);

        snr=SNo - 10*log10(fc);
        printf("\n S/N = %10.2f   dB ",snr);
        pparam->stddev=sqrt(exp(-ln(10.0)*snr/10.0));

        printf("\n Std. deviation of noise= %f\n",pparam->stddev);

        } /* end if(L2 == 1) */

```

```

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 0)    return(3);

    } /* end if(pstate == NULL) */

    if(pstate->sample_no >= pparam->samp_stop)    return(99);
    if(length_output_fifo(0) == maxlength_output_fifo(0))    return(0);

do {
    do {

        pstate->sample_temp *= 16807;
        dquo=(int)(pstate->sample_temp/2147483647.0);
        pstate->sample_temp -= dquo*2147483647.0;
        v1=2*(pstate->sample_temp/2147483647.0)-1;
        pstate->sample_temp *= 16807;
        dquo=(int)(pstate->sample_temp/2147483647.0);
        pstate->sample_temp -= dquo*2147483647.0;
        v2=2*(pstate->sample_temp/2147483647.0)-1;
        t=v1*v1+v2*v2;
    }
    while(t >= 1);

    bis=(SAMPLE)((pparam->stddev)*v1*sqrt((-2)*ln(t))/t);
    (pstate->sample_no)++;
}
while(put(0,bis) ==0) ;

    return(0);    /* normal return */
}

```

```

/***** QFDEM.C *****/
/* Function qfsk_dem() */

```

```

#include <stdio.h>
#include "type.h"
#include "star.h"

```

```

typedef struct {
    int non;
} PARAM, *PARAMPTR;

```

```

typedef struct {
    int time;
    FILE *fp;
} STATE, *STATEPTR;

```

```

qfsk_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;

```

```

{
    SAMPLE Iinp_w0,Iinp_w1,Qinp_w0,Qinp_w1,Iinp_w2,Iinp_w3,Qinp_w2,Qinp_w3;
    SAMPLE output;
    float q0,q1,q2,q3,max;
    int i,point;
    FILE *fopen();

```

```

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->fp=fopen("graf.dat","w");
        pstate->time=0;
    }

```

```

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

    while(length_input_fifo(0) > 7)
    {

```

```

        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

        get(0,&Iinp_w0);
        get(0,&Qinp_w0);
        get(0,&Iinp_w1);
        get(0,&Qinp_w1);
        get(0,&Iinp_w2);
        get(0,&Qinp_w2);
        get(0,&Iinp_w3);
        get(0,&Qinp_w3);
        q0=Iinp_w0*Iinp_w0+Qinp_w0*Qinp_w0;
        q1=Iinp_w1*Iinp_w1+Qinp_w1*Qinp_w1;
        q2=Iinp_w2*Iinp_w2+Qinp_w2*Qinp_w2;
        q3=Iinp_w3*Iinp_w3+Qinp_w3*Qinp_w3;

```

```

        max=q0; point=0;
    }

```

```

if(q1>max) {max=q1; point=1;}
if(q2>max) {max=q2; point=2;}
if(q3>max) {max=q3; point=3;}

if(pstate->time<2000)
{
    if(point==0) fprintf(pstate->fp,"\n%f, %f\r",Iinp_w0,Qinp_w0);
else if(point==1) fprintf(pstate->fp,"\n%f, %f\r",Iinp_w1,Qinp_w1);
else if(point==2) fprintf(pstate->fp,"\n%f, %f\r",Iinp_w2,Qinp_w2);
else if(point==3) fprintf(pstate->fp,"\n%f, %f\r",Iinp_w3,Qinp_w3);
}
if(pstate->time==2000) fclose(pstate->fp);

if((point&01)==1) output=1; else output= -1;
put(0,output);
if(((point>>1)&01)==1) output=1; else output= -1;
put(0,output);

pstate->time++;
}

return (0) ;
}

```

```

/***** QFMOD.C *****/
/* Function qfsk_mod() */

```

```

#include <stdio.h>
#include "type.h"
#include "star.h"
#include <mth.h>
#define PI 3.1415926535

```

```

typedef struct {
    int non;
} PARAM, *PARAMPTR;

```

```

typedef struct {
    double seed;
} STATE, *STATEPTR;

```

```

qfsk_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;

```

```

{
    SAMPLE input;
    SAMPLE Iout_w0,Qout_w0,Iout_w1,Qout_w1,Iout_w2,Qout_w2,Iout_w3,Qout_w3;
    double dquo,theta;
    int i;
    int point;

```

```

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->seed=7;
    }

```

```

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

    while(length_input_fifo(0) > 1)

```

```

    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

        get(0,&input);
        point=(input>0);
        get(0,&input);
        point += 2*(input>0);

```

```

        pstate->seed *= 16807;
        dquo=(int) (pstate->seed/2147483647.0);
        pstate->seed -= dquo*2147483647.0;
        theta=2*PI*pstate->seed/2147483647.0;

```

```

        if(point==0) {Iout_w0=cos(theta); Qout_w0=sin(theta);
                     Iout_w1=0;           Qout_w1=0;
                     Iout_w2=0;           Qout_w2=0;
                     Iout_w3=0;           Qout_w3=0;

```

```

    }
else if(point==1) {Iout_w1=cos(theta); Qout_w1=sin(theta);
                  Iout_w0=0;           Qout_w0=0;
                  Iout_w2=0;           Qout_w2=0;
                  Iout_w3=0;           Qout_w3=0;
    }
else if(point==2) {Iout_w2=cos(theta); Qout_w2=sin(theta);
                  Iout_w1=0;           Qout_w1=0;
                  Iout_w0=0;           Qout_w0=0;
                  Iout_w3=0;           Qout_w3=0;
    }
else if(point==3) {Iout_w3=cos(theta); Qout_w3=sin(theta);
                  Iout_w1=0;           Qout_w1=0;
                  Iout_w2=0;           Qout_w2=0;
                  Iout_w0=0;           Qout_w0=0;
    }

put(0,Iout_w0);
put(0,Qout_w0);
put(0,Iout_w1);
put(0,Qout_w1);
put(0,Iout_w2);
put(0,Qout_w2);
put(0,Iout_w3);
put(0,Qout_w3);

}

return (0) ;
}

```

```

/***** QPDEM.C *****/
/* Function qpsk_dem() */

#include <stdio.h>
#include "type.h"
#include "star.h"

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int time;
    FILE *fp;
} STATE, *STATEPTR;

qpsk_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE Iinput,Qinput,output;
    FILE *fopen();

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
        pstate->time=0;
        pstate->fp=fopen("graf.da+", "w");
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 1)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&Iinput);
        if(Iinput>0) output=1; else output= -1;
        put(0,output);
        get(0,&Qinput);
        if(Qinput>0) output=1; else output= -1;
        put(0,output);

        /* Print Constellation Points to File */

        if(pstate->time<2000) fprintf(pstate->fp,"\n%f, %f\r",Iinput,Qinput);
        if(pstate->time==2000) fclose(pstate->fp);
        pstate->time++;
    }

    return (0) ;
}

```

```

/***** QPMOD.C *****/
/* Function qpsk_mod() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define SCALE 0.70710678

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

qpsk_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,Ioutput,Qoutput;

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 1)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&input);
        Ioutput=SCALE*input;
        put(0,Ioutput);
        get(0,&input);
        Qoutput=SCALE*input;
        put(0,Qoutput);
    }

    return (0) ;
}

```

```

/***** RCBUT.C *****/
/* Function rclpf() */
/* File Parameter Version of Butterworth Filter */
/* Receiver Filter Application */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>
#define PI 3.1415926535
#define L 40

typedef struct {
    int none;
} PARAM,*PARAMPTR;

typedef struct {
    double a0[21], a1[21], a2[21], b1[21], b2[21], s[21], s_old1[21];
    int N;
    int time;
    FILE *finp, *fout;
} STATE,*STATEPTR;

rclpf(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE x;
    float temp;
    double tmp,Ap,Aa,fc,fs,fp,fa,wa,wc,wp;
    double d1,d2, gamma[21], s_old[21], s_old2[21];
    int i,k,option,T1;
    char *calloc();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));
    }

/* Filter Specifications */

    f1=fopen("rcvfltr1.tdt","r");
    f2=fopen("rcvfltr2.tdt","r");
    fscanf(f1,"%f",&temp); /* Read Option Type */
    option=(int)temp;
    f3=fopen("rcfilt.dat","w");
    f4=fopen("rcdelay.dat","w");

    if (option == 2) {
/* Option 2 */
/* Read in Filter Specifications */

        fscanf(f2,"%lf",&fs);
        fscanf(f2,"%lf",&fp);

```

```
fscanf(f2,"%lf",&fa);
fscanf(f2,"%lf",&Ap);
fscanf(f2,"%lf",&Aa);
```

```
printf("\n \n BUTTERWORTH RECEIVER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);
```

```
fprintf(f3,"\n BUTTERWORTH RECEIVER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);
```

```
printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");
```

```
/* Calculating Filter Parameters from Specifications */
```

```
tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
tmp=PI*fa/fs; wa=2*sin(tmp)/cos(tmp);
```

```
d1=exp(ln(10.0)*Ap/10);
d2=exp(ln(10.0)*Aa/10);
tmp=ln((d1-1)/(d2-1))/(2*ln(wp/wa));
for(pstate->N=1;pstate->N<100;pstate->N++) if(pstate->N>=tmp) break;
wc=wa/exp((1/(2.0*pstate->N))*ln(d2-1));
```

```
printf("\n Order = %2d",pstate->N);
fprintf(f3,"\n Order = %2d \r",pstate->N);
```

```
/*printf("\n wc = %10.5f rad ",wc);*/
```

```
} /* end if option == 2 */
```

```
if (option == 1) {
/* Option 1 */
/* Read in Filter Specifications */
```

```
fscanf(f1,"%f",&temp);
pstate->N=(int)temp;
fscanf(f1,"%lf",&fs);
fscanf(f1,"%lf",&fc);
```

```
printf("\n \n BUTTERWORTH RECEIVER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Order = %2d",pstate->N);
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n 3dB Cutoff Frequency = %10.2f Hz ",fc);
```

```

fprintf(f3, "\n BUTTERWORTH RECEIVER FILTER \n \r");
fprintf(f3, "\n USER FILTER SPECIFICATIONS \r");
fprintf(f3, "\n Order = %2d \r", pstate->N);
fprintf(f3, "\n Sampling Frequency = %10.2f Hz \r", fs);
fprintf(f3, "\n 3dB Cutoff Frequency = %10.2f Hz \r", fc);

```

```

/* Calculating Filter Parameters from Specifications */

```

```

tmp=PI*fc/fs; wc=2*sin(tmp)/cos(tmp);

```

```

/*printf("\n wc = %10.5f    rad ",wc);*/

```

```

} /* end if option == 1 */

```

```

if (pstate->N >= 40) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

/*
pstate->a0=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s_old1=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old2=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old=(double *) calloc(1+pstate->N/2,sizeof(double));
gamma=(double *) calloc(1+pstate->N/2,sizeof(double));
*/

```

```

/* Calculating Transfer Function Coefficients */

```

```

if(pstate->N%2==1) /* Filter Order N is odd */
{
    for(k=pstate->N+1;k<=1.5*pstate->N+1;k++)
gamma[k-pstate->N-1]=wc*cos((k-1)*PI/pstate->N);
} /* end odd N */
else /* Filter Order N is even */
{
    for(k=pstate->N+1;k<=(3*pstate->N+1)/2;k++)
gamma[k-pstate->N]=wc*cos((2*k-1)*PI/(2*pstate->N));
} /* end even N */

```

```

/*printf("\n \n Filter Coefficients:");*/
fprintf(f3, "\n \n Filter Coefficients: \r");

```

```

/* Generate the Filter Coefficients for the First Order Section */

```

```

if(pstate->N%2==1) {
pstate->a0[0]= wc/(2+wc);

```

```

pstate->a1[0]= pstate->a0[0];
pstate->b1[0]= -(2-wc)/(2+wc);
pstate->s[0]=0;

/*
printf("\n \n a0[0] = %10.5e ",pstate->a0[0]);
printf("\n a1[0] = %10.5e ",pstate->a1[0]);
printf("\n b1[0] = %10.5e ",pstate->b1[0]);
*/

fprintf(f3,"\n \n a0[0] = %10.5e \r",pstate->a0[0]);
fprintf(f3,"\n a1[0] = %10.5e \r",pstate->a1[0]);
fprintf(f3,"\n b1[0] = %10.5e \r",pstate->b1[0]);

    /*scanf("%d",&T1);*/ /* Pause */
} /* end if pstate->N%2==1 */

/* Generate the Filter Coefficients for the Second Order Sections */

for(i=1;i<=pstate->N/2;i++)
{
tmp = 4-4*gamma[i]+wc*wc;
pstate->a0[i]= wc*wc/tmp;
pstate->a1[i]= 2*pstate->a0[i];
pstate->a2[i]= pstate->a0[i];
pstate->b1[i]= -2*(4-wc*wc)/tmp;
pstate->b2[i]= (4+4*gamma[i]+wc*wc)/tmp;
pstate->s[i]=0;
pstate->s_old1[i]=0;

/*
printf("\n \n a0[%d] = %10.5e ",i,pstate->a0[i]);
printf("\n a1[%d] = %10.5e ",i,pstate->a1[i]);
printf("\n a2[%d] = %10.5e ",i,pstate->a2[i]);
printf("\n b1[%d] = %10.5e ",i,pstate->b1[i]);
printf("\n b2[%d] = %10.5e ",i,pstate->b2[i]);
*/

fprintf(f3,"\n \n a0[%d] = %10.5e \r",i,pstate->a0[i]);
fprintf(f3,"\n a1[%d] = %10.5e \r",i,pstate->a1[i]);
fprintf(f3,"\n a2[%d] = %10.5e \r",i,pstate->a2[i]);
fprintf(f3,"\n b1[%d] = %10.5e \r",i,pstate->b1[i]);
fprintf(f3,"\n b2[%d] = %10.5e \r",i,pstate->b2[i]);

    /*scanf("%d",&T1);*/ /* Pause */
}

fprintf(f4,"%3d \n \r",pstate->N); /* print filter delay in file */

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 1)    return(3);

pstate->time=0;
pstate->finp=fopen("rcinp.dat","w"); /* filter input data file */
pstate->fout=fopen("rcout.dat","w"); /* filter output data file */

```

```

    fclose(f1);
    fclose(f2);
    fclose(f3);
    fclose(f4);
    } /* end if (pstate == NULL) */

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

while(length_input_fifo(0) >0)
{
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

    get(0,&x);

    /* Write Filter Input Data to File */

    if(pstate->time<4*L)
        fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time,x);
    if(pstate->time==4*L-1) fclose(pstate->finp);

    /* Generate Filter Output */

    if(pstate->N%2==1)
    {
        s_old[0]=pstate->s[0];
        pstate->s[0]=x-pstate->b1[0]*s_old[0];
        x=pstate->a0[0]*pstate->s[0]+pstate->a1[0]*s_old[0];
    }

    if(pstate->N>1)
        for(i=1;i<=pstate->N/2;i++)
        {
            s_old2[i]=pstate->s_old1[i];
            pstate->s_old1[i]=pstate->s[i];
            pstate->s[i]=x-pstate->b1[i]*pstate->s_old1[i]-pstate->b2[i]*s_old2[i];
            x=pstate->a0[i]*pstate->s[i]+pstate->a1[i]*pstate->s_old1[i]
            +pstate->a2[i]*s_old2[i];
        }

    put(0,x);

    /* Write Filter Output Data to File */

    if(pstate->time<4*L)
        fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time,x);
    if(pstate->time==4*L-1) fclose(pstate->fout);

    pstate->time++;

}
return(0);
}

```

```

/*****          RCCHEB.C          *****/
/* Function rclpf() */
/* File Parameter Version of Chebyshev Filter */
/* Receiver Filter Application */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>
#define PI 3.1415926535
#define L 40

typedef struct {
    int none;
} PARAM, *PARAMPTR;

typedef struct {
    double a0[21], a1[21], a2[21], b1[21], b2[21], s[21], s_old1[21];
    double Ho;
    int N;
    int time;
    FILE *finp, *fout;
} STATE, *STATEPTR;

rclpf(pparam, size, pstate, pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE x;
    float temp;
    double eps, tmp, Ap, Aa, fc, fs, fp, fa, wa, wp;
    double d1, d2, s_old[21], s_old2[21];
    double sigma[83], omega[83], p0, Re_p[42], Im_p[42], eta[42], gamma[42];
    int i, k, option, T1;
    char *calloc();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL) {
        pstate=(STATEPTR) alloc_state_var(1, sizeof(STATE));
    }

/* Filter Specifications */

    f1=fopen("rcvfltr1.tdt", "r");
    f2=fopen("rcvfltr2.tdt", "r");
    fscanf(f1, "%f", &temp);
    option=(int)temp;
    f3=fopen("rcfilt.dat", "w");
    f4=fopen("rcdelay.dat", "w");

/* Read Option Type */

    if (option == 2) {
/* Option 2 */
/* Read in Filter Specifications */

```

```
fscanf(f2,"%lf",&fs);
fscanf(f2,"%lf",&fp);
fscanf(f2,"%lf",&fa);
fscanf(f2,"%lf",&Ap);
fscanf(f2,"%lf",&Aa);
```

```
printf("\n \n CHEBYCHEV RECEIVER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);
```

```
fprintf(f3,"\n CHEBYCHEV RECEIVER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);
```

```
printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");
```

```
/* Calculating Filter Parameters from Specifications */
```

```
tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
tmp=PI*fa/fs; wa=2*sin(tmp)/cos(tmp);
eps=sqrt(exp(ln(10.0)*Ap/10)-1);
d1=sqrt(exp(ln(10.0)*Aa/10)-1)/eps;
d2=wa/wp;
tmp=ln(d1+sqrt(d1*d1-1))/ln(d2+sqrt(d2*d2-1));
for(pstate->N=1;pstate->N<100;pstate->N++) if(pstate->N>=tmp) break;
```

```
printf("\n Order = %2d",pstate->N);
fprintf(f3,"\n Order = %2d \r",pstate->N);
```

```
/*printf("\n wp = %10.5f rad ",wp);
printf("\n epsilon = %10.5f ",eps);*/
```

```
} /* end if option == 2 */
```

```
if (option == 1) {
/* Option 1 */
/* Read in Filter Specifications */
```

```
fscanf(f1,"%f",&temp);
pstate->N=(int)temp;
fscanf(f1,"%lf",&fs);
fscanf(f1,"%lf",&fp);
```

```
printf("\n \n CHEBYCHEV RECEIVER FILTER \n");
```

```

printf("\n USER FILTER SPECIFICATIONS");
printf("\n Order = %2d",pstate->N);
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n 3dB Cutoff Frequency = %10.2f Hz ",fp);

fprintf(f3,"\n CHEBYCHEV RECEIVER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Order = %2d \r",pstate->N);
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n 3dB Cutoff Frequency = %10.2f Hz \r",fp);

```

/* Calculating Filter Parameters from Specifications */

```

tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
Ap=3.0;
eps=sqrt(exp(ln(10.0)*Ap/10)-1);

```

```

/*printf("\n wp = %10.5f rad ",wp);
printf("\n epsilon = %10.5f ",eps);*/

```

} /* end if option == 1 */

/*scanf("%d",&T1);*/ /* Pause */

```

if (pstate->N >= 40) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

```

/*

```

pstate->a0=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s_old1=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old2=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old=(double *) calloc(1+pstate->N/2,sizeof(double));
sigma=(double *) calloc(1+2*pstate->N,sizeof(double));
omega=(double *) calloc(1+2*pstate->N,sizeof(double));
Re_p=(double *) calloc(1+pstate->N,sizeof(double));
Im_p=(double *) calloc(1+pstate->N,sizeof(double));
eta=(double *) calloc(1+pstate->N,sizeof(double));
gamma=(double *) calloc(1+pstate->N,sizeof(double));
*/

```

/* Calculating Transfer Function Coefficients */

```

tmp=1/eps;
for(k=1;k<=2*pstate->N;k++)
{
    sigma[k]=
    -sinh(ln(tmp+sqrt(tmp*tmp+1)))/pstate->N*sin((2*k-1)*PI/(2*pstate->N));
    omega[k]=
    cosh(ln(tmp+sqrt(tmp*tmp+1)))/pstate->N*cos((2*k-1)*PI/(2*pstate->N));
}

```

```

/*printf("\n sigma(%2d) = %10.5e, omega(%2d) = %10.5e ",k,sig-
ma[k],k,omega[k]);*/
}

/*printf("\n wp = %10.5f    rad ",wp);*/

i=1;
for(k=1;k<=2*pstate->N;k++) if(sigma[k]<0)
{Re_p[i]=sigma[k];
 Im_p[i]=omega[k];
 gamma[i]=wp*Re_p[i];
 eta[i]=wp*wp*(Re_p[i]*Re_p[i]+Im_p[i]*Im_p[i]);
 i++;}

/*for(i=1;i<=pstate->N/2;i++) {
 printf("\n Re_p(%2d) = %10.5e, Im_p(%2d) = %10.5e ",i,Re_p[i],i,Im_p[i]);
 printf("\n gamma(%2d) = %10.5e, eta(%2d) = %10.5e ",i,gamma[i],i,eta[i]);
 }
*/

if(pstate->N%2==1)
{p0=sigma[(pstate->N+1)/2];

 /*printf("\n for odd N, p[0] = %10.5e ",p0);*/

 pstate->Ho= -p0;}
else
 pstate->Ho= exp(-ln(10.0)*Ap/20);

for(i=1;i<=pstate->N/2;i++)
 pstate->Ho *= Re_p[i]*Re_p[i]+Im_p[i]*Im_p[i];

 /*scanf("%d",&T1);*/ /* Pause */

/*printf("\n \n Filter Coefficients:");*/
fprintf(f3,"\n \n Filter Coefficients: \r");

/* Generate the Filter Coefficients for the First Order Section */

if(pstate->N%2==1) {
 pstate->a0[0]= wp/(2-wp*p0);
 pstate->a1[0]= pstate->a0[0];
 pstate->b1[0]= -(2+wp*p0)/(2-wp*p0);
 pstate->s[0]=0;

/*
printf("\n \n a0[0] = %10.5e ",pstate->a0[0]);
printf("\n a1[0] = %10.5e ",pstate->a1[0]);
printf("\n b1[0] = %10.5e ",pstate->b1[0]);
*/

fprintf(f3,"\n \n a0[0] = %10.5e \r",pstate->a0[0]);
fprintf(f3,"\n a1[0] = %10.5e \r",pstate->a1[0]);

```

```

fprintf(f3, "\n b1[0] = %10.5e \r", pstate->b1[0]);

    /*scanf("%d",&T1);*/ /* Pause */
} /* end if pstate->N%2==1 */

/* Generate the Filter Coefficients for the Second Order Sections */

for(i=1;i<=pstate->N/2;i++)
{
    /*printf("\n gamma(%2d) = %10.5e, eta(%2d) = %10.5e ",i,gamma[i],i,eta[i]);*/
    tmp = 4-4*gamma[i]+eta[i];
    /*printf("\ tmp = %10.5e ",tmp);*/
    pstate->a0[i]= wp*wp/tmp;
    pstate->a1[i]= 2*pstate->a0[i];
    pstate->a2[i]= pstate->a0[i];
    pstate->b1[i]= -2*(4-eta[i])/tmp;
    pstate->b2[i]= (4+4*gamma[i]+eta[i])/tmp;
    pstate->s[i]=0;
    pstate->s_old1[i]=0;

    /*
    printf("\n \n a0[%d] = %10.5e ",i,pstate->a0[i]);
    printf("\n a1[%d] = %10.5e ",i,pstate->a1[i]);
    printf("\n a2[%d] = %10.5e ",i,pstate->a2[i]);
    printf("\n b1[%d] = %10.5e ",i,pstate->b1[i]);
    printf("\n b2[%d] = %10.5e ",i,pstate->b2[i]);
    */

    fprintf(f3, "\n \n a0[%d] = %10.5e \r",i,pstate->a0[i]);
    fprintf(f3, "\n a1[%d] = %10.5e \r",i,pstate->a1[i]);
    fprintf(f3, "\n a2[%d] = %10.5e \r",i,pstate->a2[i]);
    fprintf(f3, "\n b1[%d] = %10.5e \r",i,pstate->b1[i]);
    fprintf(f3, "\n b2[%d] = %10.5e \r",i,pstate->b2[i]);

    /*scanf("%d",&T1);*/ /* Pause */
}

/*printf("\n Ho = %10.5f ",pstate->Ho);*/
fprintf(f3, "\n \n Ho = %10.5f \r",pstate->Ho);

fprintf(f4, "%3d \n \r", pstate->N); /* print filter delay in file */

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 1)    return(3);

    pstate->time=0;
    pstate->finp=fopen("rcinp.dat","w"); /* filter input data file */
    pstate->fout=fopen("rcout.dat","w"); /* filter output data file */
    fclose(f1);
    fclose(f2);
    fclose(f3);
    fclose(f4);

```

```

    } /* end if (pstate == NULL) */

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

while(length_input_fifo(0) >0)
{
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

    get(0,&x);

/* Write Filter Input Data to File */

if(pstate->time<4*L)
fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time,x);
if(pstate->time==4*L-1) fclose(pstate->finp);

/* Generate Filter Output */

x *= pstate->Ho;

if(pstate->N%2==1)
{
    s_old[0]=pstate->s[0];
    pstate->s[0]=x-pstate->b1[0]*s_old[0];
    x=pstate->a0[0]*pstate->s[0]+pstate->a1[0]*s_old[0];
}

if(pstate->N>1)
for(i=1;i<=pstate->N/2;i++)
{
    s_old2[i]=pstate->s_old1[i];
    pstate->s_old1[i]=pstate->s[i];
    pstate->s[i]=x-pstate->b1[i]*pstate->s_old1[i]-pstate->b2[i]*s_old2[i];
    x=pstate->a0[i]*pstate->s[i]+pstate->a1[i]*pstate->s_old1[i]
    +pstate->a2[i]*s_old2[i];
}

put(0,x);

/* Write Filter Output Data to File */

if(pstate->time<4*L)
fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time,x);
if(pstate->time==4*L-1) fclose(pstate->fout);

pstate->time++;
}
return(0);
}

```

```

/***** RCELL.C *****/
/* Function rclpf() */
/* File Parameter Version of Elliptic Filter */
/* Receiver Filter Application */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>
#define PI 3.1415926535
#define L 40

typedef struct {
    int none;
} PARAM,*PARAMPTR;

typedef struct {
    double a0[21], a1[21], a2[21], b1[21], b2[21], s[21], s_old1[21];
    double Ho;
    int r,N;
    int time;
    FILE *finp, *fout;
} STATE,*STATEPTR;

rclpf(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE x;
    float temp;
    double tmp,Ap,Aa,fs,fp,fa,wa,wp,Aahat;
    double d1,d2, s_old[21], s_old2[21];
    double A0[21], B0[21], B1[21], V[21], OMEGA[21];
    double W,k,kp,q0,q4,q,D,LAMBDA,lambda,slambda,sigma0,ssigma0,mu;
    int i,m,option,T1;
    char *calloc();
    double rcpow();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));
    }

/* Filter Specifications */

    f1=fopen("rcvfltr1.tdt","r");
    f2=fopen("rcvfltr2.tdt","r");
    fscanf(f1,"%f",&temp); /* Read Option Type */
    option=(int)temp;
    f3=fopen("rcfilt.dat","w");
    f4=fopen("rcdelay.dat","w");

    if (option == 2) {

```

```

/* Option 2 */
/* Read in Filter Specifications */

```

```

fscanf(f2,"%lf",&fs);
fscanf(f2,"%lf",&fp);
fscanf(f2,"%lf",&fa);
fscanf(f2,"%lf",&Ap);
fscanf(f2,"%lf",&Aa);

```

```

printf("\n \n ELLIPTIC RECEIVER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);

```

```

fprintf(f3,"\n ELLIPTIC RECEIVER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);

```

```

printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");

```

```

/* Calculating Filter Parameters from Specifications */

```

```

tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
tmp=PI*fa/fs; wa=2*sin(tmp)/cos(tmp);
k=wp/wa;

```

```

/*printf("\n Filter Selectivity k = %10.5f ",k);*/

```

```

kp=sqrt(1-k*k);
q0=0.5*(1-sqrt(kp))/(1+sqrt(kp));
q4=q0*q0*q0*q0;
q=q0+2*q0*q4+15*q0*q4*q4+150*q0*q4*q4*q4;
d1=exp(ln(10.0)*Ap/10);
d2=exp(ln(10.0)*Aa/10);
D=(d2-1)/(d1-1);
tmp=ln(16*D)/ln(1/q);
for(pstate->N=1;pstate->N<100;pstate->N++) if(pstate->N>=tmp) break;

```

```

printf("\n Filter Order N = %2d ",pstate->N);
fprintf(f3,"\n Order = %2d \r",pstate->N);

```

```

if (pstate->N >= 40) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);
}

```

```

Aahat=10.0*log10((d1-1)/(16.0*rcpow(q,pstate->N))+1);

```

```
printf("\n Actual Stopband Attenuation = %10.2f dB ",Aahat);
fprintf(f3,"\n Actual Stopband Attenuation = %10.2f dB \r",Aahat);
```

```
pstate->r=pstate->N/2;
lambda=sqrt(k)/wp;
d1=exp(ln(10.0)*Ap/20);
LAMBDA=(0.5/pstate->N)*ln((d1+1)/(d1-1));
```

```
for(m=0,d1=0;m<=4;m++)
    d1 += rcpow(-1.0,m)*rcpow(q,m*(m+1))*sinh((2*m+1)*LAMBDA);
```

```
for(m=1,d2=0;m<=5;m++)
    d2 += rcpow(-1.0,m)*rcpow(q,m*m)*cosh(2*m*LAMBDA);
```

```
sigma0=2*sqrt(sqrt(q))*d1/(1+2*d2);
if(sigma0 < 0) sigma0 = -sigma0;
ssigma0=sigma0*sigma0;
W=sqrt((1+k*ssigma0)*(1+ssigma0/k));
```

```
/*scanf("%d",&T1);*/ /* Pause */
```

```
/*
pstate->a0=(double *) calloc(1+pstate->r,sizeof(double));
pstate->a1=(double *) calloc(1+pstate->r,sizeof(double));
pstate->a2=(double *) calloc(1+pstate->r,sizeof(double));
pstate->b1=(double *) calloc(1+pstate->r,sizeof(double));
pstate->b2=(double *) calloc(1+pstate->r,sizeof(double));
pstate->s=(double *) calloc(1+pstate->r,sizeof(double));
pstate->s_old1=(double *) calloc(1+pstate->r,sizeof(double));
s_old2=(double *) calloc(1+pstate->r,sizeof(double));
s_old=(double *) calloc(1+pstate->r,sizeof(double));
A0=(double *) calloc(1+pstate->r,sizeof(double));
B0=(double *) calloc(1+pstate->r,sizeof(double));
B1=(double *) calloc(1+pstate->r,sizeof(double));
V=(double *) calloc(1+pstate->r,sizeof(double));
OMEGA=(double *) calloc(1+pstate->r,sizeof(double));
*/
```

```
/*printf("\n \n Filter Coefficients:");*/
fprintf(f3,"\n \n Filter Coefficients: \r");
```

```
/* Generate the Filter Coefficients for the First Order Section */
```

```
if(pstate->N%2==1) {
    tmp=sigma0/lambda;
    pstate->a0[0]= 1/((2+tmp)*lambda);
    pstate->a1[0]= pstate->a0[0];
    pstate->b1[0]= -(2-tmp)/(2+tmp);
    pstate->s[0]=0;
    /*printf("\n sigma0 = %10.5e ",sigma0);*/
```

```
/*
printf("\n \n a0[0] = %10.5e ",pstate->a0[0]);
printf("\n \n a1[0] = %10.5e ",pstate->a1[0]);
```

```
printf("\n b1[0] = %10.5e ",pstate->b1[0]);
*/
```

```
fprintf(f3,"\n \n a0[0] = %10.5e \r",pstate->a0[0]);
fprintf(f3,"\n a1[0] = %10.5e \r",pstate->a1[0]);
fprintf(f3,"\n b1[0] = %10.5e \r",pstate->b1[0]);
```

```
/*scanf("%d",&T1);*/ /* Pause */
```

```
} /* end if pstate->N%2==1 */
```

```
/* Generate the Filter Coefficients for the Second Order Sections */
```

```
slambda=lambda*lambda;
```

```
for(i=1;i<=pstate->r;i++)
```

```
{
/*printf("\n i= %2d ",i);*/
```

```
if(pstate->N%2==1) mu=i; else mu=i-0.5;
```

```
for(m=0,d1=0;m<=4;m++)
```

```
d1 += rcpow(-1.0,m)*rcpow(q,m*(m+1))*sin((2*m+1)*PI*mu/pstate->N);
```

```
for(m=1,d2=0;m<=5;m++)
```

```
d2 += rcpow(-1.0,m)*rcpow(q,m*m)*cos(2*m*PI*mu/pstate->N);
```

```
OMEGA[i]=2*sqrt(sqrt(q))*d1/(1+2*d2);
```

```
V[i]=sqrt((1-k*OMEGA[i]*OMEGA[i])*(1-OMEGA[i]*OMEGA[i]/k));
```

```
A0[i]=1/(OMEGA[i]*OMEGA[i]);
```

```
tmp=1+ssigma0/A0[i];
```

```
B0[i]=(ssigma0*V[i]*V[i]+W*W/A0[i])/(tmp*tmp);
```

```
B1[i]=2*sigma0*V[i]/tmp;
```

```
tmp = 4+2*B1[i]/lambda+B0[i]/slambda;
```

```
pstate->a0[i]= (4+A0[i]/slambda)/tmp;
```

```
pstate->a1[i]= -2*(4-A0[i]/slambda)/tmp;
```

```
pstate->a2[i]= pstate->a0[i];
```

```
pstate->b1[i]= -2*(4-B0[i]/slambda)/tmp;
```

```
pstate->b2[i]= (4-2*B1[i]/lambda+B0[i]/slambda)/tmp;
```

```
pstate->s[i]=0;
```

```
pstate->s_old1[i]=0;
```

```
/* Print Out Filter Coefficients */
```

```
/*
```

```
printf("\n \n A0[%d] = %10.5e ",i,A0[i]);
```

```
printf("\n B0[%d] = %10.5e ",i,B0[i]);
```

```
printf("\n B1[%d] = %10.5e ",i,B1[i]);
```

```
printf("\n a0[%d] = %10.5e ",i,pstate->a0[i]);
```

```
printf("\n a1[%d] = %10.5e ",i,pstate->a1[i]);
```

```
printf("\n a2[%d] = %10.5e ",i,pstate->a2[i]);
```

```
printf("\n b1[%d] = %10.5e ",i,pstate->b1[i]);
```

```
printf("\n b2[%d] = %10.5e ",i,pstate->b2[i]);
```

```
*/
```

```

fprintf(f3, "\n \n a0[%d] = %10.5e \r", i, pstate->a0[i]);
fprintf(f3, "\n a1[%d] = %10.5e \r", i, pstate->a1[i]);
fprintf(f3, "\n a2[%d] = %10.5e \r", i, pstate->a2[i]);
fprintf(f3, "\n b1[%d] = %10.5e \r", i, pstate->b1[i]);
fprintf(f3, "\n b2[%d] = %10.5e \r", i, pstate->b2[i]);

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

}

```

```

/* Calculate the Gain Ho */

```

```

for(i=1,tmp=1.0;i<=pstate->r;i++)
    tmp *= B0[i]/A0[i];

```

```

if(pstate->N%2==1)
    pstate->Ho=sigma0*tmp;
else
    pstate->Ho=tmp*exp(-ln(10.0)*Ap/20);

```

```

/* Print out Ho */

```

```

/*printf("\n \n Ho = %10.5f ",pstate->Ho);*/
fprintf(f3, "\n \n Ho = %10.5f \r", pstate->Ho);

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

} /* end if option == 2 */

```

```

/* Error Declarations */

```

```

if (option == 1) {
    printf("\n OPTION ERROR");
    return(2);}

```

```

fprintf(f4, "%3d \n \r", pstate->N); /* print filter delay in file */

```

```

if(no_output_fifos() != 1)    return(2);
if(no_input_fifos() != 1)    return(3);

```

```

fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);

```

```

pstate->time=0;
pstate->finp=fopen("rcinp.dat","w"); /* filter input data file */
pstate->fout=fopen("rcout.dat","w"); /* filter output data file */

```

```

}/* end if (pstate==NULL) */

```

```

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

```

```

while(length_input_fifo(0) >0)

```

```

{
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

    get(0,&x);

/* Write Filter Input Data to File */

    if(pstate->time<4*L)
        fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time,x);
    if(pstate->time==4*L-1) fclose(pstate->finp);

/* Generate Filter Output */

    x *= pstate->Ho;

    if(pstate->N%2==1)
    {
        s_old[0]=pstate->s[0];
        pstate->s[0]=x-pstate->b1[0]*s_old[0];
        x=pstate->a0[0]*pstate->s[0]+pstate->a1[0]*s_old[0];
    }

    if(pstate->N>1)
        for(i=1;i<=pstate->r;i++)
        {
            s_old2[i]=pstate->s_old1[i];
            pstate->s_old1[i]=pstate->s[i];
            pstate->s[i]=x-pstate->b1[i]*pstate->s_old1[i]-pstate->b2[i]*s_old2[i];
            x=pstate->a0[i]*pstate->s[i]+pstate->a1[i]*pstate->s_old1[i]
            +pstate->a2[i]*s_old2[i];
        }

    put(0,x);

/* Write Filter Output Data to File */

    if(pstate->time<4*L)
        fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time,x);
    if(pstate->time==4*L-1) fclose(pstate->fout);

    pstate->time++;

}
return(0);
}

/*****
double rcpow(x,i)
double x;
int i;
{
    int n;
    double p;
    if(i==0)

```

```
p=1.0;  
else  
  for(n=1,p=1;n<=i;n++)  
    p *= x;  
return(p);  
}
```

```

/*****          RCFIR.C          *****/
/* Function rclpf() */
/* File Parameter Version of FIR Filter */
/* Receiver Filter Application */

# include "type.h"
# include "star.h"
# include <stdio.h>
# include <mth.h>
# define PI 3.1415926535
# define L 40

typedef struct {
    int none;
} PARAM,*PARAMPTR;

typedef struct {
    double inp[321], h[321], Ho;
    int NN;
    int time;
    FILE *finp, *fout;
} STATE,*STATEPTR;

rclpf(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE input,output;
    double RCBESSEL();
    double tmp,tmp1,tmp2,fs,Ap,Aa,fp,fa,wa,wc,wp,d,d1,d2,D,alpha,Aahat;
    float temp;
    int i,N,n,T,T1;
    char *calloc();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));

        f1=fopen("rcvfltr1.td_", "r");
        f2=fopen("rcvfltr2.tdt", "r");
        fscanf(f1,"%f",&temp);          /* Read Option Type */
        T=(int)temp;
        f3=fopen("rcfilt.dat","w");
        f4=fopen("rcdelay.dat","w");

        if (T == 2) {
/* Option 2 */
/* Read in Filter Specifications */

            fscanf(f2,"%lf",&fs);
            fscanf(f2,"%lf",&fp);
            fscanf(f2,"%lf",&fa);
            fscanf(f2,"%lf",&Ap);
            fscanf(f2,"%lf",&Aa);

```

```

printf("\n \n FIR RECEIVER FILTER \n");
fprintf(f3,"\n FIR RECEIVER FILTER \n \r");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);

fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);

printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");

```

```

/* Calculating Filter Parameters from Specifications */

```

```

wp=2*PI*fp/fs;
wa=2*PI*fa/fs;
wc=(wa+wp)/2;
d1=exp(-ln(10.0)*Aa/20);
d=exp(ln(10.0)*Ap/20);
d2=(d-1)/(d+1);
if(d1<d2) d=d1; else d=d2;
Aahat= - 20*log10(d);
pstate->Ho=1/(1+d);

```

```

printf("\n Actual Stopband Attenuation = %10.2f dB ",Aahat);
fprintf(f3,"\n Actual Stopband Attenuation = %10.2f dB \r",Aahat);

```

```

/*printf("\n Gain Adjustment = %10.2f ",pstate->Ho);*/

```

```

if(Aahat<=21) {
alpha=0;
D=0.9222;}
else if (Aahat>50)
alpha=0.1102*(Aahat-8.7);
else
alpha=0.5842*exp(0.4*ln(Aahat-21))+0.07886*(Aahat-21);

```

```

if(Aahat>21)
D=(Aahat-7.95)/14.36;

```

```

tmp=1+D*2*PI/(wa-wp);

```

```

for(N=1;N<1000;N += 2) if(N>=tmp) break;

```

```

printf("\n Filter Order = %3d ",N);
fprintf(f3,"\n Filter Order = %3d \r",N);

```

```

/*printf("\n alpha = %10.2f ",alpha);*/
/*printf("\n D = %10f ",D);*/

```

```

if (N>=320) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

/* Calculating the Filter Coefficients */

```

```

pstate->NN=(N-1)/2;

```

```

/* pstate->inp = (double *) calloc(2*pstate->NN+1,sizeof(double));
pstate->h = (double *) calloc(2*pstate->NN+1,sizeof(double));
*/

```

```

/*printf("\n \n Filter Impulse Response: \n");*/
fprintf(f3,"\n \n Filter Impulse Response: \n \r");

```

```

for(n= -pstate->NN;n<=pstate->NN;n++)
{
    if(n==0)
        tmp=wc/PI;
    else
        tmp=sin(n*wc)/(PI*n);

```

```

tmp1=(n*1.0)/pstate->NN;
tmp2=RCBESSEL(alpha*sqrt(1-tmp1*tmp1))/RCBESSEL(alpha);

```

```

pstate->h[n+pstate->NN]=tmp*tmp2;

```

```

/*printf("\n h[%d] = %10.5e ",n+pstate->NN,pstate->h[n+pstate->NN]);*/
fprintf(f3,"\n h[%d] = %10.5e \r",n+pstate->NN,pstate->h[n+pstate->NN]);
}

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

} /* end if (T == 2), Option 2 */

```

```

if (T == 1) {
/* Option 1 - Read in the Filter Order and Sampling Frequency */

```

```

fscanf(f1,"%f",&temp);
N=(int)temp;
fscanf(f1,"%lf",&fs);
fscanf(f1,"%lf",&fp);

```

```

printf("\n \n FIR RECEIVER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Filter Order = %3d ",N);

```

```

printf("\n Sampling Frequency = %10.2f  Hz ",fs);
printf("\n Cutoff Frequency = %10.2f Hz",fp);
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Filter Order = %3d \r",N);
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Cutoff Frequency = %10.2f Hz \r",fp);

if (N>=320) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

/* Read in the Filter Coefficients */

pstate->NN=(N-1)/2;

/* pstate->inp = (double *) calloc(2*pstate->NN+1,sizeof(double));
pstate->h    = (double *) calloc(2*pstate->NN+1,sizeof(double));
*/

/*printf("\n \n Filter Impulse Response: \n");*/
fprintf(f3,"\n \n Filter Impulse Response: \n \r");

for(n= -pstate->NN;n<=pstate->NN;n++)
{
    fscanf(f1,"%lf",&pstate->h[n+pstate->NN]);
    /*printf("\n h[%d] = %10.5e ",n+pstate->NN,pstate->h[n+pstate->NN]);*/
    fprintf(f3,"\n h[%d] = %10.5e \r",n+pstate->NN,pstate->h[n+pstate->NN]);
} /* end for */

pstate->Ho=1;

/*scanf("%d",&T1);*/ /* Pause */

} /* end if (T == 1) - Option 1 */

for(i=2*pstate->NN-1;i>=0;i--) pstate->inp[i]=0;

fprintf(f4,"%3d \n \r",pstate->NN); /*print filter delay in rcdelay.dat*/

if(no_output_fifos() != 1) return(2);
if(no_input_fifos() != 1) return(3);

pstate->time=0;
pstate->finp=fopen("rcinp.dat","w"); /* filter input data file */
pstate->fout=fopen("rcout.dat","w"); /* filter output data file */
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
} /* end if (pstate == NULL */

```

```

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

while(length_input_fifo(0) >0)
{
if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

get(0,&input);

/* Write Filter Input Data to File */
if(pstate->time<4*L)
fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time, input);
if(pstate->time==4*L-1) fclose(pstate->finp);

/* Perform Convolution to Generate Filter Output */

input *= pstate->Ho;
for(i=2*pstate->NN-1;i>=0;i--) pstate->inp[i+1]=pstate->inp[i];
pstate->inp[0]=input;
output=0;
for(n= -pstate->NN;n<=pstate->NN;n++)
output += pstate->h[n+pstate->NN]*pstate->inp[n+pstate->NN];
put(0,output);

/* Write Filter Output Data to File */
if(pstate->time<4*L)
fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time, output);
if(pstate->time==4*L-1) fclose(pstate->fout);

pstate->time++;
}
return(0);
}

/*****
double RCBESSEL(x)
double x;
{
double I0,t,t2;
t=x/3.75;
t2=t*t;
if(x>=0 && x<=3.75)
I0=1+t2*(3.5156229+t2*(3.0899424+t2*(1.2067492+t2*(0.2659732
+t2*(0.0360768+t2*0.0045813)))));
else if(x>3.75)
I0=(exp(x)/sqrt(x))*(0.39894228+(0.01328592+(0.00225319
+(-0.00157565+(0.00916281+(-0.02057706
+(0.02635537+(-0.01647633+0.00392377/t)/t)/t)/t)/t)/t)/t);
return(I0);
}

```

```

/***** RSCOD.C *****/
/***** (15,9) Reed-Solomon Encoder *****/
/* Function rscod() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define TT 6
#define N 15
#define K 9
#define CNT pstate->cnt
#define VV pstate->vv
#define G pstate->g
#define B pstate->b
#define H pstate->h
#define F pstate->f
typedef struct {
    int non;
}
PARAM, *PARAMPTR;
typedef struct {
    unsigned char b[TT], g[TT+1], cnt, h[N], f[N+1], vv;
}
STATE, *STATEPTR;

rscod (pparam, size, pstate, pstar)
PARAMPTR pparam;
STATEPTR pstate;
STARPTR pstar;
int size;
{
    SAMPLE bit;
    int i, j, symbol;

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=2)
            return(3);
    }

    /*** H[ ] and F[ ] compute the power and log in GF(16) *****/

    H[0]=1;
    for(i=0; i<4; i++) H[i+1]=2*H[i];
    for(i=4; i<N; i++) H[i]=H[i-3]^H[i-4];

    for(j=1; j<N+1; j++) {
        for(i=0; i<N; i++) {
            if(H[i]==j) F[j]=i;
        }
    }

    F[0]=0;
    CNT=0;
    VV=0;

```

```

    /*** g[ ] are the coefficients of the generating polynomial ***/

```

```

    G[0]=H[6];
    G[1]=H[9];
    G[2]=H[6];
    G[3]=H[4];
    G[4]=H[14];
    G[5]=H[10];
    G[6]=H[0];

```

```

    )

```

```

    /*
    if(length_output_fifo(0) != length_output_fifo(1)) return(7);
    */
    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
    if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);

```

```

    while(length_input_fifo(0) >0 || CNT>=K )
    {

```

```

    /*
    printf("\ninp rscod0=%4d CNT=%4d",length_input_fifo(0),CNT);
    */

```

```

    if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

    /*
    if(length_output_fifo(1)==maxlength_output_fifo(1)) return(0);
    */

```

```

    if(CNT==0) for(i=0;i<TT;i++) B[i]=0;

```

```

    /*** information bits ***/

```

```

    if(CNT<K)
    {

```

```

        symbol=0;
        for(i=0;i<4;i++)
        {
            get(0,&bit) ;
            put(1,bit);
            symbol ^= ((int)bit)<<i;
        }
        VV=(symbol)^B[TT-1];
    }

```

```

    else /*** parity bits ***/
    {

```

```

        symbol=B[TT-1];
        VV=0;
    }

```

```

    for(i=TT-1;i>0;i--)
    B[i]=B[i-1]^(VV!=0)*(H[(F[G[i]]+F[VV])%N]);
    B[0]=(VV!=0)*(H[(F[G[0]]+F[VV])%N]);

```

```

    /***

```

```

        for(i=0;i<4;i++)
        {
            bit=(symbol>>i)&01;
            put(0,bit);

```

```
}
```

```
CNT=(CNT+1)%N;
```

```
}
```

```
/*
```

```
printf("\nout rscod0=%4d",length_output_fifo(0));
```

```
printf("\nout rscod1=%4d",length_output_fifo(1));
```

```
*/
```

```
return (0) ;
```

```
}
```

```

/***** RSDEC.C *****/
/**** (15,9) Reed-Solomon Decoder *****/
/* Function rsdec() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define TT 6
#define N 15
#define K 9
#define H pstate->h
#define F pstate->f
typedef struct {
    int non;
}
PARAM, *PARAMPTR;
typedef struct {
    unsigned char h[N], f[N+1];
}
STATE, *STATEPTR;

rsdec (pparam, size, pstate, pstar)
PARAMPTR pparam;
STATEPTR pstate;
STARPTR pstar;
int size;
{
    SAMPLE bit;
    unsigned char e[N+1], E[N+1], S[TT+1], sig[TT+1], degR, degQ, a, b, tem, Q[TT+1];
    unsigned char R[TT+1], mu[TT+1], lam[TT+1], REC[N], TEM[TT+1];
    int i, j, L, CL, TH, symbol;

    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

/**** H[ ] and F[ ] compute the power and log in GF(16) *****/

    H[0]=1;
    for(i=0;i<4;i++) H[i+1]=2*H[i];
    for(i=4;i<N;i++) H[i]=H[i-3]^H[i-4];

    for(j=1;j<N+1;j++) {
        for(i=0;i<N;i++) {
            if(H[i]==j) F[j]=i;
        }
    }

    F[0]=99;

}

if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

```

```

while(length_input_fifo(0) >0 )
{
    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
/*****
for(j=0;j<N;j++)
{
    symbol=0;
    for(i=0;i<4;i++) {get(0,&bit); symbol ^= ((int)bit)<<i;}
    REC[j]=(char)symbol;
}

/***** Syndrome calculation *****/
for(j=0;j<=TT;j++) S[j]=0;
for(i=0;i<N;i++)
{
    for(j=0;j<TT;j++) S[j]=REC[i]^((S[j] != 0)*(H[(6-j+F[S[j]])%N]));
}
/*
for(j=0;j<TT;j++) printf("\n   %4d%4d%4d",j,S[j],F[S[j]]);
*/

for(j=0;j<TT;j++)
{
    S[j]=0;
    for(i=0;i<N;i++) S[j] ^= ((REC[N-1-i] != 0)*(H[(i*(j+1)+F[REC[N-1-i]])%N]));
}
/*
for(j=0;j<TT;j++) printf("\n %4d%4d%4d",j,S[j],F[S[j]]);
*/

/***** Modified Euclid Algorithm *****/
for(j=0;j<=TT;j++) {R[j]=0; lam[j]=0; mu[j]=0; E[j]=S[TT-j]; }
for(j=0;j<=TT;j++) {R[j]=0; lam[j]=0; mu[j]=0; E[j+1]=S[j]; Q[j]=0;}
for(j=0;j<TT;j++) Q[j]=S[TT-1-j];
R[TT]=1;
mu[0]=1;
degR=TT;
degQ=TT;
i=1;
TH=TT/2;
while(i<=TT)
{
    j=degR;
    while(R[j]==0) degR= --j;
    if(degR>TT) degR=0;
    j=degQ;
    while(Q[j]==0) degQ= --j;
    if(degQ>TT) degQ=0;

    L=degR-degQ;
    CL=L;
    if(L<0) CL= -L;
}

```

```

/*
printf("\n\n i=%3d",i);
printf("\n degR=%3d degQ=%3d L=%3d ",degR,degQ,L);
*/
/*
for(j=0;j<=TT;j++)
{
printf("\n Q%3d %3d %3d ",j,Q[j],F[Q[j]]);
printf("    R%3d %3d ",R[j],F[R[j]]);
}
*/

if(degR<TH || degQ<TH)
{
if(degR<TH)    { for(j=0;j<=TT;j++) sig[j]=lam[j]; }
else          { for(j=0;j<=TT;j++) sig[j]=mu[j]; }
i=TT+1;
}
else
{
if(degR<degQ)
{
for(j=0;j<=TT;j++)
{
TEM[j]=R[j];
R[j]=Q[j];
Q[j]=TEM[j];
}
for(j=0;j<=TT;j++)
{
TEM[j]=lam[j];
lam[j]=mu[j];
mu[j]=TEM[j];
}
tem=degR;
degR=degQ;
degQ=tem;
}
if(Q[degQ]==0)
{
degQ--;
if(degQ<TH)
{
for(j=0;j<=TT;j++) sig[j]=mu[j];
i=TT+1;
}
}
else
{
degR--;
/* compute R lam *****/
a=R[degR+1];
b=Q[degQ];
/*
printf("\n a=%3d b=%3d",F[a],F[b]);

```

```

*/
for(j=0;j<=TT;j++)
{
    TEM[j]=(j>=CL)*Q[j-CL];
    R[j]=(R[j]!=0)*(b!=0)*(H[(F[b]+F[R[j]])%N])^(TEM[j]!=0)
    *(a!=0)*(H[(F[a]+F[TEM[j]])%N]);
    TEM[j]=(j>=CL)*mu[j-CL];
    lam[j]=(lam[j]!=0)*(b!=0)*(H[(F[b]+F[lam[j]])%N])^(TEM[j]!=0)
    *(a!=0)*(H[(F[a]+F[TEM[j]])%N]);
}
/*****/

    if(degR<TH)
    {
        for(j=0;j<=TT;j++) sig[j]=lam[j];
        i=TT+1;
    }
}
/*
    for(j=0;j<=TT;j++)
    {
        printf("\n  Q%3d %3d ",Q[j],F[Q[j]]);
        printf("    R%3d %3d ",R[j],F[R[j]]);
        printf("    lam%3d %3d ",lam[j],F[lam[j]]);
        printf("    mu%3d %3d ",mu[j],F[mu[j]]);
    }

*/
    }
    i++;
}
/***** Error locator polynomial *****/
degR=TT;
j=degR;
while(sig[j]==0) degR= --j;
if(degR>TT) degR=0;

tem=sig[degR];
for(j=0;j<=TT;j++)
{
    sig[j]=(sig[j]!=0)*(H[(N-F[tem]+F[sig[j]])%N]);
}
/*
    printf("\n  sig%3d %3d %3d ",j,sig[j],F[sig[j]]);
*/
}
/*****/
for(j=TT-1;j<=N-2;j++)
{
    E[j+2]=0;
    for(i=0;i<degR;i++)
    {
        tem=sig[degR-i-1];
        E[j+2] ^= (tem!=0)*(E[j+1-i]!=0)*(H[(F[tem]+F[E[j+1-i]])%N]);
    }
}
E[0]=E[N];

```

```

for(j=0;j<=N;j++)
/*
printf("\n  E%3d %3d %3d ",j,E[j],F[E[j]]);
*/

/*****Inverse transform *****/
for(j=0;j<=N;j++) e[j]=0;
for(i=0;i<N;i++)
{
for(j=0;j<N;j++) e[j]=E[N-i-1]^((e[j] != 0)*(H[(j+1+F[e[j]])%N]));
}

for(j=0;j<N;j++)
{
e[j]=0;
for(i=0;i<N;i++) e[j] ^= ((E[i] !=0)*(H[(N*N-i*(j))+F[E[i]])%N]));
}

for(j=0;j<K;j++)
{
symbol=REC[j]^e[N-1-j];

for(i=0;i<4;i++) (bit=(symbol>>i)&01; put(0,bit);)
}
}
return (0);
}

```

```

/***** SIN.C *****/
/* Function gen() */
/* Generates a Sinusoidal Signal of Given Frequency */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <math.h>
#define PI 3.1415926535

typedef struct {
    int sample_stp;
    int seed;
} PARAM, *PARAMPTR;

typedef struct {
    int sample_no;
    int sample_stop;
    float tmp;
} STATE, *STATEPTR;

gen(pparam,size,pstate,pstar)

PARAMPTR pparam;
int size;
STATEPTR pstate;
STARPTR pstar;
{
    SAMPLE x;
    FILE *fopen(), *f1;
    float fs, fc;

    if(pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        pstate->sample_no = 0;
        if(size == 0) pstate->sample_stop = 100;
        else if(size == sizeof(PARAM))
            pstate->sample_stop = pparam->sample_stp;
        else return(1);
        if(no_output_fifos() != 1) return(2);
        if(no_input_fifos() != 0) return(3);

        f1=fopen("source.tdt","r");
        fscanf(f1,"%f",&fs);
        fscanf(f1,"%f",&fc);
        fclose(f1);

        printf("\n Sampling Frequency = %10.2f Hz",fs);
        printf("\n Sinusoidal Frequency = %10.2f Hz",fc);

        pstate->tmp = (2*PI*fc)/fs;
    }

    if(pstate->sample_no >= pparam->sample_stp) return(99);
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

```

```

while(length_output_fifo(0)<maxlength_output_fifo(0))
{
    x = sin(pstate->tmp*pstate->sample_no);

    /*printf("\n %10d %10.4f",pstate->sample_no,x);*/

    put(0,x);
    pstate->sample_no++;
}
    return(0);
}

```

```

/***** SINK.C *****/
/* Function sink() */

#include "type.h"
#include "star.h"
#include <stdio.h>

typedef struct {
    int sample_stp;
    int seed;
} PARAM, *PARAMPTR;

typedef struct {
    float time; /* holds current time, which is printed */
    float time_scale;
} STATE, *STATEPTR;

sink(pparam,size,pstate,pstar)

PARAMPTR pparam;
int size; /* size of parameter storage */
STATEPTR pstate;
STARPTR pstar;
{
    SAMPLE x;
    int i,j,c;
    float disp;

    if(pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if(no_input_fifos() != 1) return(1);
        if(no_output_fifos() != 0) return(2);
        pstate->time_scale = 1.0;
    }

    if(size<=0) return(204);

    pstate->time = 0.0;
}

while(length_input_fifo(0)>0)
{
    get(0,&x);
    /*printf("\n%10d %10.3f", (int)pstate->time,x);*/
    pstate->time += 1;
}

return(0);
}

```

```

/***** TBPDEM.C *****/
/* Function bpsk_dem() */
/* Time Domain Version */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define L 40

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

bpsk_dem (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,output;
    float sum;
    int i;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        sum=0;
        for(i=0;i<L;i++)
        {
            get(0,&input);
            /*sum += input;*/
            if(i==(L/2 - 1)) sum = input;
        }

        if(sum>0) output=1; else output= -1;
        put(0,output);
    }

    return (0) ;
}

```

```

/***** TBPMOD.C *****/
/* Function bpsk_mod() */
/* Time Domain Version */

#include <stdio.h>
#include <mth.h>
#include "type.h"
#include "star.h"
#define L 40

typedef struct {
    int non;
} PARAM, *PARAMPTR;

typedef struct {
    int none;
} STATE, *STATEPTR;

bpsk_mod (pparam,size,pstate,pstar)
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
    int size;
{
    SAMPLE input,output;
    int i;
    if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);
    }

    if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

    while(length_input_fifo(0) > 0)
    {
        if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
        get(0,&input);
        /*output=input/sqrt((float)L);*/
        output=input;
        for(i=0;i<L;i++) put(0,output);
    }

    return (0) ;
}

```

```

/***** TXBUT.C *****/
/* Function txlpf() */
/* File Parameter Version of Butterworth Filter */
/* Transmitter Filter Application */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>
#define PI 3.1415926535
#define L 40

typedef struct {
    int none;
} PARAM,*PARAMPTR;

typedef struct {
    double a0[21], a1[21], a2[21], b1[21], b2[21], s[21], s_old1[21];
    int N;
    int time;
    FILE *finp, *fout;
} STATE,*STATEPTR;

txlpf(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE x;
    float temp;
    double tmp,Ap,Aa,fc,fs,fp,fa,wa,wc,wp;
    double d1,d2, gamma[21], s_old[21], s_old2[21];
    int i,k,option,T1;
    char *calloc();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));
    }

/* Filter Specifications */

    f1=fopen("chlfltr1.tdt","r");
    f2=fopen("chlfltr2.tdt","r");
    fscanf(f1,"%f",&temp); /* Read Option Type */
    option=(int)temp;
    f3=fopen("txfilt.dat","w");
    f4=fopen("txdelay.dat","w");

    if (option == 2) {
/* Option 2 */
/* Read in Filter Specifications */

        fscanf(f2,"%lf",&fs);
        fscanf(f2,"%lf",&fp);

```

```
fscanf(f2,"%lf",&fa);
fscanf(f2,"%lf",&Ap);
fscanf(f2,"%lf",&Aa);
```

```
printf("\n \n BUTTERWORTH TRANSMITTER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);
```

```
fprintf(f3,"\n BUTTERWORTH TRANSMITTER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);
```

```
printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");
```

```
/* Calculating Filter Parameters from Specifications */
```

```
tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
tmp=PI*fa/fs; wa=2*sin(tmp)/cos(tmp);
```

```
d1=exp(ln(10.0)*Ap/10);
d2=exp(ln(10.0)*Aa/10);
tmp=ln((d1-1)/(d2-1))/(2*ln(wp/wa));
for(pstate->N=1;pstate->N<100;pstate->N++) if(pstate->N>=tmp) break;
wc=wa/exp((1/(2.0*pstate->N))*ln(d2-1));
```

```
printf("\n Order = %2d",pstate->N);
fprintf(f3,"\n Order = %2d \r",pstate->N);
```

```
/*printf("\n wc = %10.5f rad ",wc);*/
```

```
} /* end if option == 2 */
```

```
if (option == 1) {
```

```
/* Option 1 */
```

```
/* Read in Filter Specifications */
```

```
fscanf(f1,"%f",&temp);
pstate->N=(int)temp;
fscanf(f1,"%lf",&fs);
fscanf(f1,"%lf",&fc);
```

```
printf("\n \n BUTTERWORTH TRANSMITTER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Order = %2d",pstate->N);
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n 3dB Cutoff Frequency = %10.2f Hz ",fc);
```

```

    fprintf(f3, "\n BUTTERWORTH TRANSMITTER FILTER \n \r");
    fprintf(f3, "\n USER FILTER SPECIFICATIONS \r");
    fprintf(f3, "\n Order = %2d \r", pstate->N);
    fprintf(f3, "\n Sampling Frequency = %10.2f Hz \r", fs);
    fprintf(f3, "\n 3dB Cutoff Frequency = %10.2f Hz \r", fc);

```

```

/* Calculating Filter Parameters from Specifications */

```

```

    tmp=PI*fc/fs; wc=2*sin(tmp)/cos(tmp);

```

```

    /*printf("\n wc = %10.5f    rad ", wc);*/

```

```

} /* end if option == 1 */

```

```

if (pstate->N >= 40) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);
}

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

/*
pstate->a0=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s_old1=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old2=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old=(double *) calloc(1+pstate->N/2,sizeof(double));
gamma=(double *) calloc(1+pstate->N/2,sizeof(double));
*/

```

```

/* Calculating Transfer Function Coefficients */

```

```

if(pstate->N%2==1) /* Filter Order N is odd */
{
    for(k=pstate->N+1;k<=1.5*pstate->N+1;k++)
        gamma[k-pstate->N-1]=wc*cos((k-1)*PI/pstate->N);
} /* end odd N */
else /* Filter Order N is even */
{
    for(k=pstate->N+1;k<=(3*pstate->N+1)/2;k++)
        gamma[k-pstate->N]=wc*cos((2*k-1)*PI/(2*pstate->N));
} /* end even N */

```

```

/*printf("\n \n Filter Coefficients:");*/
fprintf(f3, "\n \n Filter Coefficients: \r");

```

```

/* Generate the Filter Coefficients for the First Order Section */

```

```

if(pstate->N%2==1) {
    pstate->a0[0]= wc/(2+wc);
}

```

```

pstate->a1[0]= pstate->a0[0];
pstate->b1[0]= -(2-wc)/(2+wc);
pstate->s[0]=0;

/*
printf("\n \n a0[0] = %10.5e ",pstate->a0[0]);
printf("\n \n a1[0] = %10.5e ",pstate->a1[0]);
printf("\n \n b1[0] = %10.5e ",pstate->b1[0]);
*/

fprintf(f3,"\n \n a0[0] = %10.5e \r",pstate->a0[0]);
fprintf(f3,"\n \n a1[0] = %10.5e \r",pstate->a1[0]);
fprintf(f3,"\n \n b1[0] = %10.5e \r",pstate->b1[0]);

    /*scanf("%d",&T1);*/ /* Pause */
} /* end if pstate->N%2==1 */

/* Generate the Filter Coefficients for the Second Order Sections */

for(i=1;i<=pstate->N/2;i++)
{
tmp = 4-4*gamma[i]+wc*wc;
pstate->a0[i]= wc*wc/tmp;
pstate->a1[i]= 2*pstate->a0[i];
pstate->a2[i]= pstate->a0[i];
pstate->b1[i]= -2*(4-wc*wc)/tmp;
pstate->b2[i]= (4+4*gamma[i]+wc*wc)/tmp;
pstate->s[i]=0;
pstate->s_old1[i]=0;

/*
printf("\n \n a0[%d] = %10.5e ",i,pstate->a0[i]);
printf("\n \n a1[%d] = %10.5e ",i,pstate->a1[i]);
printf("\n \n a2[%d] = %10.5e ",i,pstate->a2[i]);
printf("\n \n b1[%d] = %10.5e ",i,pstate->b1[i]);
printf("\n \n b2[%d] = %10.5e ",i,pstate->b2[i]);
*/

fprintf(f3,"\n \n a0[%d] = %10.5e \r",i,pstate->a0[i]);
fprintf(f3,"\n \n a1[%d] = %10.5e \r",i,pstate->a1[i]);
fprintf(f3,"\n \n a2[%d] = %10.5e \r",i,pstate->a2[i]);
fprintf(f3,"\n \n b1[%d] = %10.5e \r",i,pstate->b1[i]);
fprintf(f3,"\n \n b2[%d] = %10.5e \r",i,pstate->b2[i]);

    /*scanf("%d",&T1);*/ /* Pause */
}

fprintf(f4,"%3d \n \r",pstate->N); /* print filter delay in file */

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 1)    return(3);

pstate->time=0;
pstate->finp=fopen("txinp.dat","w"); /* filter input data file */
pstate->fout=fopen("txout.dat","w"); /* filter output data file */

```

```

    fclose(f1);
    fclose(f2);
    fclose(f3);
    fclose(f4);
    } /* end if (pstate == NULL) */

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

while(length_input_fifo(0) >0)
{
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

    get(0,&x);

    /* Write Filter Input Data to File */

    if(pstate->time<4*L)
        fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time,x);
    if(pstate->time==4*L-1) fclose(pstate->finp);

    /* Generate Filter Output */

    if(pstate->N%2==1)
    {
        s_old[0]=pstate->s[0];
        pstate->s[0]=x-pstate->b1[0]*s_old[0];
        x=pstate->a0[0]*pstate->s[0]+pstate->a1[0]*s_old[0];
    }

    if(pstate->N>1)
        for(i=1;i<=pstate->N/2;i++)
        {
            s_old2[i]=pstate->s_old1[i];
            pstate->s_old1[i]=pstate->s[i];
            pstate->s[i]=x-pstate->b1[i]*pstate->s_old1[i]-pstate->b2[i]*s_old2[i];
            x=pstate->a0[i]*pstate->s[i]+pstate->a1[i]*pstate->s_old1[i]
            +pstate->a2[i]*s_old2[i];
        }

    put(0,x);

    /* Write Filter Output Data to File */

    if(pstate->time<4*L)
        fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time,x);
    if(pstate->time==4*L-1) fclose(pstate->fout);

    pstate->time++;

}
return(0);
}

```

```

/***** TXCHEB.C *****/
/* Function txlpf() */
/* File Parameter Version of Chebychev Filter */
/* Transmitter Filter Application */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>
#define PI 3.1415926535
#define L 40

typedef struct {
    int none;
} PARAM,*PARAMPTR;

typedef struct {
    double a0[21], a1[21], a2[21], b1[21], b2[21], s[21], s_old1[21];
    double Ho;
    int N;
    int time;
    FILE *finp, *fout;
} STATE,*STATEPTR;

txlpf(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE x;
    float temp;
    double eps,tmp,Ap,Aa,fc,fs,fp,fa,wa,wp;
    double d1,d2, s_old[21], s_old2[21];
    double sigma[83], omega[83], p0, Re_p[42], Im_p[42], eta[2], gamma[42];
    int i,k,option,T1;
    char *calloc();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));
    }

/* Filter Specifications */

    f1=fopen("ch1fltr1.tdt","r");
    f2=fopen("ch1fltr2.tdt","r");
    fscanf(f1,"%f",&temp); /* Read Option Type */
    option=(int)temp;
    f3=fopen("txfilt.dat","w");
    f4=fopen("txdelay.dat","w");

    if (option == 2) {
/* Option 2 */
/* Read in Filter Specifications */

```

```

fscanf(f2,"%lf",&fs);
fscanf(f2,"%lf",&fp);
fscanf(f2,"%lf",&fa);
fscanf(f2,"%lf",&Ap);
fscanf(f2,"%lf",&Aa);

```

```

printf("\n \n CHEBYCHEV TRANSMITTER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);

```

```

fprintf(f3,"\n CHEBYCHEV TRANSMITTER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);

```

```

printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");

```

```

/* Calculating Filter Parameters from Specifications */

```

```

tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
tmp=PI*fa/fs; wa=2*sin(tmp)/cos(tmp);
eps=sqrt(exp(ln(10.0)*Ap/10)-1);
d1=sqrt(exp(ln(10.0)*Aa/10)-1)/eps;
d2=wa/wp;
tmp=ln(d1+sqrt(d1*d1-1))/ln(d2+sqrt(d2*d2-1));
for(pstate->N=1;pstate->N<100;pstate->N++) if(pstate->N>=tmp) break;

```

```

printf("\n Order = %2d",pstate->N);
fprintf(f3,"\n Order = %2d \r",pstate->N);

```

```

/*printf("\n wp = %10.5f rad ",wp);
printf("\n epsilon = %10.5f ",eps);*/

```

```

} /* end if option == 2 */

```

```

if (option == 1) {
/* Option 1 */
/* Read in Filter Specifications */

```

```

fscanf(f1,"%f",&temp);
pstate->N=(int)temp;
fscanf(f1,"%lf",&fs);
fscanf(f1,"%lf",&fp);

```

```

printf("\n \n CHEBYCHEV TRANSMITTER FILTER \n");

```

```

printf("\n USER FILTER SPECIFICATIONS");
printf("\n Order = %2d",pstate->N);
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n 3dB Cutoff Frequency = %10.2f Hz ",fp);

fprintf(f3,"\n CHEBYCHEV TRANSMITTER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Order = %2d \r",pstate->N);
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n 3dB Cutoff Frequency = %10.2f Hz \r",fp);

```

/* Calculating Filter Parameters from Specifications */

```

tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
Ap=3.0;
eps=sqrt(exp(ln(10.0)*Ap/10)-1);

```

```

/*printf("\n wp = %10.5f rad ",wp);
printf("\n epsilon = %10.5f ",eps);*/

```

} /* end if option == 1 */

/*scanf("%d",&T1);*/ /* Pause */

```

if (pstate->N >= 40) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

```

```

/*
pstate->a0=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->a2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b1=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->b2=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s=(double *) calloc(1+pstate->N/2,sizeof(double));
pstate->s_old1=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old2=(double *) calloc(1+pstate->N/2,sizeof(double));
s_old=(double *) calloc(1+pstate->N/2,sizeof(double));
sigma=(double *) calloc(1+2*pstate->N,sizeof(double));
omega=(double *) calloc(1+2*pstate->N,sizeof(double));
Re_p=(double *) calloc(1+pstate->N,sizeof(double));
Im_p=(double *) calloc(1+pstate->N,sizeof(double));
eta=(double *) calloc(1+pstate->N,sizeof(double));
gamma=(double *) calloc(1+pstate->N,sizeof(double));
*/

```

/* Calculating Transfer Function Coefficients */

```

tmp=1/eps;
for(k=1;k<=2*pstate->N;k++)
{
    sigma[k]=
    -sinh(ln(tmp+sqrt(tmp*tmp+1)))/pstate->N*sin((2*k-1)*PI/(2*pstate->N));
    omega[k]=
    cosh(ln(tmp+sqrt(tmp*tmp+1)))/pstate->N*cos((2*k-1)*PI/(2*pstate->N));
}

```

```

/*printf("\n sigma(%2d) = %10.5e, omega(%2d) = %10.5e ",k,sig-
ma[k],k,omega[k]);*/
}

/*printf("\n wp = %10.5f   rad ",wp);*/

i=1;
for(k=1;k<=2*pstate->N;k++) if(sigma[k]<0)
{Re_p[i]=sigma[k];
 Im_p[i]=omega[k];
 gamma[i]=wp*Re_p[i];
 eta[i]=wp*wp*(Re_p[i]*Re_p[i]+Im_p[i]*Im_p[i]);
 i++;}

/*for(i=1;i<=pstate->N/2;i++) {
 printf("\n Re_p(%2d) = %10.5e, Im_p(%2d) = %10.5e ",i,Re_p[i],i,Im_p[i]);
 printf("\n gamma(%2d) = %10.5e, eta(%2d) = %10.5e ",i,gamma[i],i,eta[i]);
}
*/

if(pstate->N%2==1)
{p0=sigma[(pstate->N+1)/2];

 /*printf("\n for odd N, p[0] = %10.5e ",p0);*/

 pstate->Ho= -p0;}
else
 pstate->Ho= exp(-ln(10.0)*Ap/20);

for(i=1;i<=pstate->N/2;i++)
 pstate->Ho *= Re_p[i]*Re_p[i]+Im_p[i]*Im_p[i];

 /*scanf("%d",&T1);*/ /* Pause */

/*printf("\n \n Filter Coefficients:");*/
fprintf(f3,"\n \n Filter Coefficients: \r");

/* Generate the Filter Coefficients for the First Order Section */

if(pstate->N%2==1) {
 pstate->a0[0]= wp/(2-wp*p0);
 pstate->a1[0]= pstate->a0[0];
 pstate->b1[0]= -(2+wp*p0)/(2-wp*p0);
 pstate->s[0]=0;

/*
printf("\n \n a0[0] = %10.5e ",pstate->a0[0]);
printf("\n \n a1[0] = %10.5e ",pstate->a1[0]);
printf("\n \n b1[0] = %10.5e ",pstate->b1[0]);
*/

fprintf(f3,"\n \n a0[0] = %10.5e \r",pstate->a0[0]);
fprintf(f3,"\n \n a1[0] = %10.5e \r",pstate->a1[0]);

```

```

fprintf(f3, "\n b1[0] = %10.5e \r", pstate->b1[0]);

    /*scanf("%d",&T1);*/ /* Pause */
} /* end if pstate->N%2==1 */

/* Generate the Filter Coefficients for the Second Order Sections */

for(i=1;i<=pstate->N/2;i++)
{
/*printf("\n gamma(%2d) = %10.5e, eta(%2d) = %10.5e ",i,gamma[i],i,eta[i]);*/
tmp = 4-4*gamma[i]+eta[i];
/*printf("\ tmp = %10.5e ",tmp);*/
pstate->a0[i]= wp*wp/tmp;
pstate->a1[i]= 2*pstate->a0[i];
pstate->a2[i]= pstate->a0[i];
pstate->b1[i]= -2*(4-eta[i])/tmp;
pstate->b2[i]= (4+4*gamma[i]+eta[i])/tmp;
pstate->s[i]=0;
pstate->s_old1[i]=0;

/*
printf("\n \n a0[%d] = %10.5e ",i,pstate->a0[i]);
printf("\n a1[%d] = %10.5e ",i,pstate->a1[i]);
printf("\n a2[%d] = %10.5e ",i,pstate->a2[i]);
printf("\n b1[%d] = %10.5e ",i,pstate->b1[i]);
printf("\n b2[%d] = %10.5e ",i,pstate->b2[i]);
*/

fprintf(f3, "\n \n a0[%d] = %10.5e \r",i,pstate->a0[i]);
fprintf(f3, "\n a1[%d] = %10.5e \r",i,pstate->a1[i]);
fprintf(f3, "\n a2[%d] = %10.5e \r",i,pstate->a2[i]);
fprintf(f3, "\n b1[%d] = %10.5e \r",i,pstate->b1[i]);
fprintf(f3, "\n b2[%d] = %10.5e \r",i,pstate->b2[i]);

    /*scanf("%d",&T1);*/ /* Pause */
}

/*printf("\n Ho = %10.5f ",pstate->Ho);*/
fprintf(f3, "\n \n Ho = %10.5f \r",pstate->Ho);

fprintf(f4, "%3d \n \r",pstate->N); /* print filter delay in file */

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 1)    return(3);

pstate->time=0;
pstate->finp=fopen("txinp.dat","w"); /* filter input data file */
pstate->fout=fopen("txout.dat","w"); /* filter output data file */
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);

```

```

    } /* end if (pstate == NULL) */

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) < 1) return(0);

while(length_input_fifo(0) > 0)
{
    if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

    get(0,&x);

/* Write Filter Input Data to File */

if(pstate->time<4*L)
fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time,x);
if(pstate->time==4*L-1) fclose(pstate->finp);

/* Generate Filter Output */

x *= pstate->Ho;

if(pstate->N%2==1)
{
    s_old[0]=pstate->s[0];
    pstate->s[0]=x-pstate->b1[0]*s_old[0];
    x=pstate->a0[0]*pstate->s[0]+pstate->a1[0]*s_old[0];
}

if(pstate->N>1)
for(i=1;i<=pstate->N/2;i++)
{
    s_old2[i]=pstate->s_old1[i];
    pstate->s_old1[i]=pstate->s[i];
    pstate->s[i]=x-pstate->b1[i]*pstate->s_old1[i]-pstate->b2[i]*s_old2[i];
    x=pstate->a0[i]*pstate->s[i]+pstate->a1[i]*pstate->s_old1[i]
    +pstate->a2[i]*s_old2[i];
}

put(0,x);

/* Write Filter Output Data to File */

if(pstate->time<4*L)
fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time,x);
if(pstate->time==4*L-1) fclose(pstate->fout);

pstate->time++;
}
return(0);
}

```

```

/***** TXELL.C *****/
/* Function txlpf() */
/* File Parameter Version of Elliptic Filter */
/* Transmitter Filter Application */

# include "type.h"
# include "star.h"
# include <stdio.h>
# include <mth.h>
# define PI 3.1415926535
# define L 40

typedef struct {
    int none;
} PARAM,*PARAMPTR;

typedef struct {
    double a0[21], a1[21], a2[21], b1[21], b2[21], s[21], s_old1[21];
    double Ho;
    int r,N;
    int time;
    FILE *finp, *fout;
} STATE,*STATEPTR;

txlpf(pparam,size,pstate,pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE x;
    float temp;
    double tmp,Ap,Aa,fs,fp,fa,wa,wp,Aahat;
    double d1,d2, s_old[21], s_old2[21];
    double A0[21], B0[21], B1[21], V[21], OMEGA[21];
    double W,k,kp,q0,q4,q,D,LAMBDA,lambda,slambda,sigma0,ssigma0,mu;
    int i,m,option,T1;
    char *calloc();
    double txpow();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL ) {
        pstate=(STATEPTR) alloc_state_var(1,sizeof(STATE));
    }

/* Filter Specifications */

    f1=fopen("chlfltr1.tdt","r");
    f2=fopen("chlfltr2.tdt","r");
    fscanf(f1,"%f",&temp); /* Read Option Type */
    option=(int)temp;
    f3=fopen("txfilt.dat","w");
    f4=fopen("txdelay.dat","w");

    if (option == 2) {

```

```

/* Option 2 */
/* Read in Filter Specifications */

```

```

fscanf(f2,"%lf",&fs);
fscanf(f2,"%lf",&fp);
fscanf(f2,"%lf",&fa);
fscanf(f2,"%lf",&Ap);
fscanf(f2,"%lf",&Aa);

```

```

printf("\n \n ELLIPTIC TRANSMITTER FILTER \n");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);

```

```

fprintf(f3,"\n ELLIPTIC TRANSMITTER FILTER \n \r");
fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);

```

```

printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");

```

```

/* Calculating Filter Parameters from Specifications */

```

```

tmp=PI*fp/fs; wp=2*sin(tmp)/cos(tmp);
tmp=PI*fa/fs; wa=2*sin(tmp)/cos(tmp);
k=wp/wa;

```

```

/*printf("\n Filter Selectivity k = %10.5f ",k);*/

```

```

kp=sqrt(1-k*k);
q0=0.5*(1-sqrt(kp))/(1+sqrt(kp));
q4=q0*q0*q0*q0;
q=q0+2*q0*q4+15*q0*q4*q4+150*q0*q4*q4*q4;
d1=exp(ln(10.0)*Ap/10);
d2=exp(ln(10.0)*Aa/10);
D=(d2-1)/(d1-1);
tmp=ln(16*D)/ln(1/q);
for(pstate->N=1;pstate->N<100;pstate->N++) if(pstate->N>=tmp) break;

```

```

printf("\n Filter Order N = %2d ",pstate->N);
fprintf(f3,"\n Order = %2d \r",pstate->N);

```

```

if (pstate->N >= 40) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

```

```

Aahat=10.0*log10((d1-1)/(16.0*txpow(q,pstate->N))+1);

```

```

printf("\n Actual Stopband Attenuation = %10.2f dB ",Aahat);
fprintf(f3,"\n Actual Stopband Attenuation = %10.2f dB \r",Aahat);

```

```

pstate->r=pstate->N/2;
lambda=sqrt(k)/wp;
d1=exp(ln(10.0)*Ap/20);
LAMBDA=(0.5/pstate->N)*ln((d1+1)/(d1-1));

```

```

for(m=0,d1=0;m<=4;m++)
    d1 += txpow(-1.0,m)*txpow(q,m*(m+1))*sinh((2*m+1)*LAMBDA);

```

```

for(m=1,d2=0;m<=5;m++)
    d2 += txpow(-1.0,m)*txpow(q,m*m)*cosh(2*m*LAMBDA);

```

```

sigma0=2*sqrt(sqrt(q))*d1/(1+2*d2);
if(sigma0 < 0) sigma0 = -sigma0;
ssigma0=sigma0*sigma0;
W=sqrt((1+k*ssigma0)*(1+ssigma0/k));

```

```

/*scanf("%d",&T1);*/ /* Pause */

```

```

/*
pstate->a0=(double *) calloc(1+pstate->r,sizeof(double));
pstate->a1=(double *) calloc(1+pstate->r,sizeof(double));
pstate->a2=(double *) calloc(1+pstate->r,sizeof(double));
pstate->b1=(double *) calloc(1+pstate->r,sizeof(double));
pstate->b2=(double *) calloc(1+pstate->r,sizeof(double));
pstate->s=(double *) calloc(1+pstate->r,sizeof(double));
pstate->s_old1=(double *) calloc(1+pstate->r,sizeof(double));
s_old2=(double *) calloc(1+pstate->r,sizeof(double));
s_old=(double *) calloc(1+pstate->r,sizeof(double));
A0=(double *) calloc(1+pstate->r,sizeof(double));
B0=(double *) calloc(1+pstate->r,sizeof(double));
B1=(double *) calloc(1+pstate->r,sizeof(double));
V=(double *) calloc(1+pstate->r,sizeof(double));
OMEGA=(double *) calloc(1+pstate->r,sizeof(double));
*/

```

```

/*printf("\n \n Filter Coefficients:");*/
fprintf(f3,"\n \n Filter Coefficients: \r");

```

```

/* Generate the Filter Coefficients for the First Order Section */

```

```

if(pstate->N%2==1) {
    tmp=sigma0/lambda;
    pstate->a0[0]= 1/((2+tmp)*lambda);
    pstate->a1[0]= pstate->a0[0];
    pstate->b1[0]= -(2-tmp)/(2+tmp);
    pstate->s[0]=0;
    /*printf("\n sigma0 = %10.5e ",sigma0);*/

```

```

    /*
    printf("\n \n a0[0] = %10.5e ",pstate->a0[0]);
    printf("\n \n a1[0] = %10.5e ",pstate->a1[0]);

```

```
printf("\n b1[0] = %10.5e ",pstate->b1[0]);
*/
```

```
fprintf(f3,"\n \n a0[0] = %10.5e \r",pstate->a0[0]);
fprintf(f3,"\n a1[0] = %10.5e \r",pstate->a1[0]);
fprintf(f3,"\n b1[0] = %10.5e \r",pstate->b1[0]);
```

```
/*scanf("%d",&T1);*/ /* Pause */
```

```
} /* end if pstate->N%2==1 */
```

```
/* Generate the Filter Coefficients for the Second Order Sections */
```

```
slambda=lambda*lambda;
```

```
for(i=1;i<=pstate->r;i++)
```

```
{
/*printf("\n i= %2d ",i);*/
```

```
if(pstate->N%2==1) mu=i; else mu=i-0.5;
```

```
for(m=0,d1=0;m<=4;m++)
```

```
    d1 += txpow(-1.0,m)*txpow(q,m*(m+1))*sin((2*m+1)*PI*mu/pstate->N);
```

```
for(m=1,d2=0;m<=5;m++)
```

```
    d2 += txpow(-1.0,m)*txpow(q,m*m)*cos(2*m*PI*mu/pstate->N);
```

```
OMEGA[i]=2*sqrt(sqrt(q))*d1/(1+2*d2);
```

```
V[i]=sqrt((1-k*OMEGA[i]*OMEGA[i])*(1-OMEGA[i]*OMEGA[i]/k));
```

```
A0[i]=1/(OMEGA[i]*OMEGA[i]);
```

```
tmp=1+ssigma0/A0[i];
```

```
B0[i]=(ssigma0*V[i]*V[i]+W*W/A0[i])/(tmp*tmp);
```

```
B1[i]=2*sigma0*V[i]/tmp;
```

```
tmp = 4+2*B1[i]/lambda+B0[i]/slambda;
```

```
pstate->a0[i]= (4+A0[i]/slambda)/tmp;
```

```
pstate->a1[i]= -2*(4-A0[i]/slambda)/tmp;
```

```
pstate->a2[i]= pstate->a0[i];
```

```
pstate->b1[i]= -2*(4-B0[i]/slambda)/tmp;
```

```
pstate->b2[i]= (4-2*B1[i]/lambda+B0[i]/slambda)/tmp;
```

```
pstate->s[i]=0;
```

```
pstate->s_old1[i]=0;
```

```
/* Print Out Filter Coefficients */
```

```
/*
```

```
printf("\n \n A0[%d] = %10.5e ",i,A0[i]);
```

```
printf("\n B0[%d] = %10.5e ",i,B0[i]);
```

```
printf("\n B1[%d] = %10.5e ",i,B1[i]);
```

```
printf("\n a0[%d] = %10.5e ",i,pstate->a0[i]);
```

```
printf("\n a1[%d] = %10.5e ",i,pstate->a1[i]);
```

```
printf("\n a2[%d] = %10.5e ",i,pstate->a2[i]);
```

```
printf("\n b1[%d] = %10.5e ",i,pstate->b1[i]);
```

```
printf("\n b2[%d] = %10.5e ",i,pstate->b2[i]);
```

```
*/
```

```

fprintf(f3, "\n \n a0[%d] = %10.5e \r", i, pstate->a0[i]);
fprintf(f3, "\n a1[%d] = %10.5e \r", i, pstate->a1[i]);
fprintf(f3, "\n a2[%d] = %10.5e \r", i, pstate->a2[i]);
fprintf(f3, "\n b1[%d] = %10.5e \r", i, pstate->b1[i]);
fprintf(f3, "\n b2[%d] = %10.5e \r", i, pstate->b2[i]);

    /*scanf("%d",&T1);*/ /* Pause */
}

/* Calculate the Gain Ho */
for(i=1,tmp=1.0;i<=pstate->r;i++)
    tmp *= B0[i]/A0[i];

if(pstate->N%2==1)
    pstate->Ho=sigma0*tmp;
else
    pstate->Ho=tmp*exp(-ln(10.0)*Ap/20);

/* Print out Ho */

    /*printf("\n \n Ho = %10.5f ",pstate->Ho);*/
    fprintf(f3, "\n \n Ho = %10.5f \r", pstate->Ho);

    /*scanf("%d",&T1);*/ /* Pause */

} /* end if option == 2 */

/* Error Declarations */
if (option == 1) {
    printf("\n OPTION ERROR");
    return(2);}

fprintf(f4, "%3d \n \r", pstate->N); /* print filter delay in file */

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 1)    return(3);

    fclose(f1);
    fclose(f2);
    fclose(f3);
    fclose(f4);

    pstate->time=0;
    pstate->finp=fopen("txinp.dat","w"); /* filter input data file */
    pstate->fout=fopen("txout.dat","w"); /* filter output data file */

    }/* end if (pstate==NULL) */

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

while(length_input_fifo(0) >0)

```

```

    {
        if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

        get(0,&x);

        /* Write Filter Input Data to File */

        if(pstate->time<4*L)
            fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time,x);
        if(pstate->time==4*L-1) fclose(pstate->finp);

        /* Generate Filter Output */

        x *= pstate->Ho;

        if(pstate->N%2==1)
        {
            s_old[0]=pstate->s[0];
            pstate->s[0]=x-pstate->b1[0]*s_old[0];
            x=pstate->a0[0]*pstate->s[0]+pstate->a1[0]*s_old[0];
        }

        if(pstate->N>1)
            for(i=1;i<=pstate->r;i++)
            {
                s_old2[i]=pstate->s_old1[i];
                pstate->s_old1[i]=pstate->s[i];
                pstate->s[i]=x-pstate->b1[i]*pstate->s_old1[i]-pstate->b2[i]*s_old2[i];
                x=pstate->a0[i]*pstate->s[i]+pstate->a1[i]*pstate->s_old1[i]
                +pstate->a2[i]*s_old2[i];
            }

        put(0,x);

        /* Write Filter Output Data to File */

        if(pstate->time<4*L)
            fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time,x);
        if(pstate->time==4*L-1) fclose(pstate->fout);

        pstate->time++;

    }
    return(0);
}

/*****
double txpow(x,i)
double x;
int i;
{
    int n;
    double p;
    if(i==0)

```

```
    p=1.0;  
else  
    for(n=1,p=1;n<=i;n++)  
        p *= x;  
return(p);  
}
```

```

/***** TXFIR.C *****/
/* Function txlpf() */
/* File Parameter Version of FIR Filter */
/* Transmitter Filter Application */

#include "type.h"
#include "star.h"
#include <stdio.h>
#include <mth.h>
#define PI 3.1415926535
#define L 40

typedef struct {
    int none;
} PARAM, *PARAMPTR;

typedef struct {
    double inp[321], h[321], Ho;
    int NN;
    int time;
    FILE *finp, *fout;
} STATE, *STATEPTR;

txlpf(pparam, size, pstate, pstar)
    int size;
    PARAMPTR pparam;
    STATEPTR pstate;
    STARPTR pstar;
{
    SAMPLE input, output;
    double TXBESSEL();
    double tmp, tmp1, tmp2, fs, Ap, Aa, fp, fa, wa, wc, wp, d, d1, d2, D, alpha, Aahat;
    float temp;
    int i, N, n, T, T1;
    char *calloc();
    FILE *fopen(), *f1, *f2, *f3, *f4;

    if (pstate == NULL) {
        pstate=(STATEPTR) alloc_state_var(1, sizeof(STATE));

        f1=fopen("chlfltr1.tdt", "r");
        f2=fopen("chlfltr2.tdt", "r");
        fscanf(f1, "%f", &temp); /* Read Option Type */
        T=(int)temp;
        f3=fopen("txfilt.dat", "w");
        f4=fopen("txdelay.dat", "w");

        if (T == 2) {
            /* Option 2 */
            /* Read in Filter Specifications */

            fscanf(f2, "%lf", &fs);
            fscanf(f2, "%lf", &fp);
            fscanf(f2, "%lf", &fa);
            fscanf(f2, "%lf", &Ap);
            fscanf(f2, "%lf", &Aa);

```

```

printf("\n \n FIR TRANSMITTER FILTER \n");
fprintf(f3,"\n FIR TRANSMITTER FILTER \n \r");
printf("\n USER FILTER SPECIFICATIONS");
printf("\n Sampling Frequency = %10.2f Hz ",fs);
printf("\n Passband Edge Frequency = %10.2f Hz ",fp);
printf("\n Stopband Edge Frequency = %10.2f Hz ",fa);
printf("\n Maximum Passband Attenuation = %10.2f dB ",Ap);
printf("\n Minimum Stopband Attenuation = %10.2f dB ",Aa);

fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
fprintf(f3,"\n Passband Edge Frequency = %10.2f Hz \r",fp);
fprintf(f3,"\n Stopband Edge Frequency = %10.2f Hz \r",fa);
fprintf(f3,"\n Maximum Passband Attenuation = %10.2f dB \r",Ap);
fprintf(f3,"\n Minimum Stopband Attenuation = %10.2f dB \r",Aa);

printf("\n \n FILTER DESIGN RESULTS");
fprintf(f3,"\n \n FILTER DESIGN RESULTS \r");

```

```

/* Calculating Filter Parameters from Specifications */

```

```

wp=2*PI*fp/fs;
wa=2*PI*fa/fs;
wc=(wa+wp)/2;
d1=exp(-ln(10.0)*Aa/20);
d=exp(ln(10.0)*Ap/20);
d2=(d-1)/(d+1);
if(d1<d2) d=d1; else d=d2;
Aahat= - 20*log10(d);
pstate->Ho=1/(1+d);

```

```

printf("\n Actual Stopband Attenuation = %10.2f dB ",Aahat);
fprintf(f3,"\n Actual Stopband Attenuation = %10.2f dB \r",Aahat);

```

```

/*printf("\n Gain Adjustment = %10.2f ",pstate->Ho);*/

```

```

if(Aahat<=21) {
alpha=0;
D=0.9222;}
else if (Aahat>50)
alpha=0.1102*(Aahat-8.7);
else
alpha=0.5842*exp(0.4*ln(Aahat-21))+0.07886*(Aahat-21);

```

```

if(Aahat>21)
D=(Aahat-7.95)/14.36;

```

```

tmp=1+D*2*PI/(wa-wp);

```

```

for(N=1;N<1000;N += 2) if(N>=tmp) break;

```

```

printf("\n Filter Order = %3d ",N);
fprintf(f3,"\n Filter Order = %3d \r",N);

```

```

/*printf("\n alpha = %10.2f ",alpha);*/
/*printf("\n D = %10f ",D);*/

if (N>=320) {
    printf("\n FILTER ORDER OUT OF RANGE");
    return(3);}

/*scanf("%d",&T1);*/ /* Pause */

/* Calculating the Filter Coefficients */

pstate->NN=(N-1)/2;

/* pstate->inp = (double *) calloc(2*pstate->NN+1,sizeof(double));
pstate->h = (double *) calloc(2*pstate->NN+1,sizeof(double));
*/

/*printf("\n \n Filter Impulse Response: \n");*/
fprintf(f3,"\n \n Filter Impulse Response: \n \r");

for(n= -pstate->NN;n<=pstate->NN;n++)
{
    if(n==0)
        tmp=wc/PI;
    else
        tmp=sin(n*wc)/(PI*n);

    tmp1=(n*1.0)/pstate->NN;
    tmp2=TXBESSEL(alpha*sqrt(1-tmp1*tmp1))/TXBESSEL(alpha);

    pstate->h[n+pstate->NN]=tmp*tmp2;

/*printf("\n h[%d] = %10.5e ",n+pstate->NN,pstate->h[n+pstate->NN]);*/
fprintf(f3,"\n h[%d] = %10.5e \r",n+pstate->NN,pstate->h[n+pstate->NN]);

}

/*scanf("%d",&T1);*/ /* Pause */

} /* end if (T == 2), Option 2 */

if (T == 1) {
/* Option 1 - Read in the Filter Order and Sampling Frequency */

    fscanf(f1,"%f",&temp);
    N=(int)temp;
    fscanf(f1,"%lf",&fs);
    fscanf(f1,"%lf",&fp);

    printf("\n \n FIR TRANSMITTER FILTER \n");
    printf("\n USER FILTER SPECIFICATIONS");
    printf("\n Filter Order = %3d ",N);

```

```

    printf("\n Sampling Frequency = %10.2f  Hz ",fs);
    printf("\n Cutoff Frequency = %10.2f Hz",fp);
    fprintf(f3,"\n USER FILTER SPECIFICATIONS \r");
    fprintf(f3,"\n Filter Order = %3d \r",N);
    fprintf(f3,"\n Sampling Frequency = %10.2f Hz \r",fs);
    fprintf(f3,"\n Cutoff Frequency = %10.2f Hz \r",fp);

    if (N>=320) {
        printf("\n FILTER ORDER OUT OF RANGE");
        return(3);}

/* Read in the Filter Coefficients */

    pstate->NN=(N-1)/2;

/* pstate->inp = (double *) calloc(2*pstate->NN+1,sizeof(double));
   pstate->h = (double *) calloc(2*pstate->NN+1,sizeof(double));
*/

/*printf("\n \n Filter Impulse Response: \n");*/
fprintf(f3,"\n \n Filter Impulse Response: \n \r");

for(n= -pstate->NN;n<=pstate->NN;n++)
{
    fscanf(f1,"%lf",&pstate->h[n+pstate->NN]);
    /*printf("\n h[%d] = %10.5e ",n+pstate->NN,pstate->h[n+pstate->NN]);*/
    fprintf(f3,"\n h[%d] = %10.5e \r",n+pstate->NN,pstate->h[n+pstate->NN])
} /* end for */

    pstate->Ho=1;

    /*scanf("%d",&T1);*/    /* Pause */

} /* end if (T == 1) - Option 1 */

    for(i=2*pstate->NN-1;i>=0;i--) pstate->inp[i]=0;

fprintf(f4,"%3d \n \r",pstate->NN); /*print filter delay in txdelay.dat*/

    if(no_output_fifos() != 1)    return(2);
    if(no_input_fifos() != 1)    return(3);

    pstate->time=0;
    pstate->finp=fopen("txinp.dat","w"); /* filter input data file */
    pstate->fout=fopen("txout.dat","w"); /* filter output data file */
    fclose(f1);
    fclose(f2);
    fclose(f3);
    fclose(f4);
} /* end if (pstate == NULL */

```

```

if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0) <1) return(0);

while(length_input_fifo(0) >0)
{
if(length_output_fifo(0) == maxlength_output_fifo(0)) return(0);

get(0,&input);

/* Write Filter Input Data to File */
if(pstate->time<4*L)
fprintf(pstate->finp,"\n %f, %f\r", (float)pstate->time, input);
if(pstate->time==4*L-1) fclose(pstate->finp);

/* Perform Convolution to Generate Filter Output */

input *= pstate->Ho;
for(i=2*pstate->NN-1;i>=0;i--) pstate->inp[i+1]=pstate->inp[i];
pstate->inp[0]=input;
output=0;
for(n= -pstate->NN;n<=pstate->NN;n++)
output += pstate->h[n+pstate->NN]*pstate->inp[n+pstate->NN];
put(0,output);

/* Write Filter Output Data to File */
if(pstate->time<4*L)
fprintf(pstate->fout,"\n %f, %f\r", (float)pstate->time, output);
if(pstate->time==4*L-1) fclose(pstate->fout);

pstate->time++;
}
return(0);
}

/*****
double TXBESSEL(x)
double x;
{
double I0,t,t2;
t=x/3.75;
t2=t*t;
if(x>=0 && x<=3.75)
I0=1+t2*(3.5156229+t2*(3.0899424+t2*(1.2067492+t2*(0.2659732
+t2*(0.0360768+t2*0.0045813)))));
else if(x>3.75)
I0=(exp(x)/sqrt(x))*(0.39894228+(0.01328592+(0.00225319
+(-0.00157565+(0.00916281+(-0.02057706
+(0.02635537+(-0.01647633+0.00392377/t)/t)/t)/t)/t)/t)/t);
return(I0);
}
*****/

```

```

/***** VIT3.C ( VITERBI DECODER (3,1/2) ) *****/
/* Function viterbi() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define NORM      pstate->norm
#define METRIC    pstate->metric
#define SURV      pstate->surv
#define TIME      pstate->time
#define K         pstate->k
#define NS        4
#define T         32

typedef struct {
    unsigned int surv[NS];
    float metric[NS], norm;
    int k[NS], time;
} STATE, *STATEPTR;

viterbi (file_name, size, pstate, pstar)

    STATEPTR pstate;
    STARPTR pstar;
    char *file_name;
    int size;
{
    SAMPLE rec0, rec1, u;
    unsigned int j, i, h, s, t, ssurv[NS];
    float branch[4], L0, L1, mmetric[NS];

    if (pstate == NULL)
    {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if (no_input_fifos( ) != 1 || no_output_fifos( ) != 1) return(3);

        TIME=T-1;
        printf("\nTrunc. Length = %d", T);
        for(j=0; j<NS; j++)
        {
            K[j]=2*(j%2)+((j^j)>>1)%2);
            METRIC[j]=10000;
            SURV[j]=0;
        }
        METRIC[0]=0;
        NORM=0;
    }

/*****

if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0)%2 != 0) return(34);

while(length_input_fifo(0) > 1)

```

```

{
    if(length_output_fifo(0)==maxlength_output_fifo(0))return(0);
    get(0,&rec0);
    get(0,&rec1);

    branch[0]=(rec0+1)*(rec0+1)+(rec1+1)*(rec1+1);
    branch[1]=(rec0+1)*(rec0+1)+(rec1-1)*(rec1-1);
    branch[2]=(rec1+1)*(rec1+1)+(rec0-1)*(rec0-1);
    branch[3]=(rec0-1)*(rec0-1)+(rec1-1)*(rec1-1);
    /*****/

    h=TIME--;
    if(TIME<0) TIME += T;
    s=~(01<<h);

    for(j=0;j<NS;j++) {ssurv[j]=SURV[j]; mmetric[j]=METRIC[j]-NORM;}

    NORM=100000;
    for(j=0;j<NS;j++)
    {
        i=j>>1;
        L0=mmetric[i]+branch[K[j]];
        L1=mmetric[i|2]+branch[3-K[j]];
        if(L1 < L0) {METRIC[j]=L1; SURV[j]=(ssurv[(j>>1)|2]& s)|((j&01)<<h);}
        else      {METRIC[j]=L0; SURV[j]=(ssurv[j>>1]& s)|((j&01)<<h);}
        if( METRIC[j]<NORM) {NORM=METRIC[j]; t=j;}
    }

    u=(SAMPLE)((SURV[t] >>TIME)&01);
    put(0,u);
    }

    return (0) ;
}

```

```

/**** VIT3Q.C (VITERBI DECODER (3,1/2) FOR SOFT DEC QPSK) ****/
/* Function viterbi() */

```

```

#include <stdio.h>
#include "type.h"
#include "star.h"
#define NORM      pstate->norm
#define METRIC     pstate->metric
#define SURV      pstate->surv
#define TIME      pstate->time
#define K         pstate->k
#define NS        4
#define T         32
#define SCALE     0.7071068

```

```

typedef struct {
    unsigned int surv[NS];
    float metric[NS],norm;
    int k[NS],time;
} STATE, *STATEPTR;

```

```

viterbi (file_name,size,pstate,pstar)

```

```

    STATEPTR pstate;
    STARPTR pstar;
    char *file_name;
    int size;

```

```

{
    SAMPLE rec0,rec1,u;
    unsigned int j,i,h,s,t,ssurv[NS];
    float branch[4],L0,L1,mmetric[NS];

```

```

    if (pstate == NULL)
    {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);

```

```

        TIME=T-1;
        printf("\nTrunc. Length = %d",T);
        for(j=0;j<NS;j++)
        {
            K[j]=2*(j%2)+((j^j)>>1)%2);
            METRIC[j]=10000;
            SURV[j]=0;
        }
        METRIC[0]=0;
        NORM=0;
    }

```

```

/*****

```

```

if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0)%2 != 0) return(34);

```

```

while(length_input_fifo(0) > 1)
{
    if(length_output_fifo(0)==maxlength_output_fifo(0))return(0);
    get(0,&rec0);
    get(0,&rec1);
    branch[0]=(rec0+SCALE)*(rec0+SCALE)+(rec1+SCALE)*(rec1+SCALE);
    branch[1]=(rec0+SCALE)*(rec0+SCALE)+(rec1-SCALE)*(rec1-SCALE);
    branch[2]=(rec1+SCALE)*(rec1+SCALE)+(rec0-SCALE)*(rec0-SCALE);
    branch[3]=(rec0-SCALE)*(rec0-SCALE)+(rec1-SCALE)*(rec1-SCALE);
    /*****/

    h=TIME--;
    if(TIME<0) TIME += T;
    s=~(01<<h);

    for(j=0;j<NS;j++) {ssurv[j]=SURV[j]; mmetric[j]=METRIC[j]-NORM;}

    NORM=100000;
    for(j=0;j<NS;j++)
    {
        i=j>>1;
        L0=mmetric[i]+branch[K[j]];
        L1=mmetric[i|2]+branch[3-K[j]];
        if(L1 < L0) {METRIC[j]=L1; SURV[j]=(ssurv[(j>>1)|2]& s)|((j&01)<<h);}
        else      {METRIC[j]=L0; SURV[j]=(ssurv[j>>1]& s)|((j&01)<<h);}
        if( METRIC[j]<NORM) {NORM=METRIC[j]; t=j;}
    }

    u=(SAMPLE)((SURV[t] >>TIME)&01);
    put(0,u);
    return (0) ;
}

```

```

/***** VIT7.C ( VITERBI DECODER (7,1/2) ) *****/
/* Function viterbi() */

#include <stdio.h>
#include "type.h"
#include "star.h"
#define NORM      pstate->norm
#define METRIC     pstate->metric
#define SURV      pstate->surv
#define TIME      pstate->time
#define K         pstate->k
#define NS        64
#define T         32

typedef struct {
    unsigned int surv[NS];
    float metric[NS], norm;
    int k[NS], time;
} STATE, *STATEPTR;

viterbi (file_name, size, pstate, pstar)

    STATEPTR pstate;
    STARPTR pstar;
    char *file_name;
    int size;
{
    SAMPLE rec0, rec1, u;
    unsigned int j, i, h, s, t, ssurv[NS];
    float branch[4], L0, L1, mmetric[NS];

    if (pstate == NULL)
    {
        pstate = (STATEPTR) alloc_state_var(1, sizeof(STATE));
        if (no_input_fifos( ) != 1 || no_output_fifos( ) != 1) return(3);

        TIME=T-1;
        printf("\nTrunc. Length = %d", T);
        for(j=0; j<NS; j++)
        {
            K[j]=2*((j^j>>2^j>>3^j>>5)%2)+((j^j>>1^j>>2^j>>3)%2);
            METRIC[j]=10000;
            SURV[j]=0;
        }
        METRIC[0]=0;
        NORM=0;
    }

/*****

if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0)%2 != 0) return(34);

while(length_input_fifo(0) > 1)

```

```

{
    if(length_output_fifo(0)==maxlength_output_fifo(0))return(0);
    get(0,&rec0);
    get(0,&rec1);

    branch[0]=(rec0+1)*(rec0+1)+(rec1+1)*(rec1+1);
    branch[1]=(rec0+1)*(rec0+1)+(rec1-1)*(rec1-1);
    branch[2]=(rec1+1)*(rec1+1)+(rec0-1)*(rec0-1);
    branch[3]=(rec0-1)*(rec0-1)+(rec1-1)*(rec1-1);
    /*****/

    h=TIME--;
    if(TIME<0) TIME += T;
    s=~(01<<h);

    for(j=0;j<NS;j++) {ssurv[j]=SURV[j]; mmetric[j]=METRIC[j]-NORM;}

    NORM=100000;
    for(j=0;j<NS;j++)
    {
        i=j>>1;
        L0=mmetric[i]+branch[K[j]];
        L1=mmetric[i|32]+branch[3-K[j]];
        if(L1 < L0) {METRIC[j]=L1; SURV[j]=(ssurv[(j>>1)|32]& s)|((j&01)<<h);}
        else      {METRIC[j]=L0; SURV[j]=(ssurv[j>>1]& s)|((j&01)<<h);}
        if( METRIC[j]<NORM) {NORM=METRIC[j]; t=j;}
    }

    u=(SAMPLE)((SURV[t] >>TIME)&01);
    put(0,u);
    }
    return (0) ;
}

```

```

/**** VIT7Q.C ( VITERBI DECODER (7,1/2) FOR SFT DEC QPSK) ****/
/* Function viterbi() */

```

```

#include <stdio.h>
#include "type.h"
#include "star.h"
#define NORM      pstate->norm
#define METRIC     pstate->metric
#define SURV      pstate->surv
#define TIME      pstate->time
#define K         pstate->k
#define NS        64
#define T         32
#define SCALE     0.7071068

```

```

typedef struct {
    unsigned int surv[NS];
    float metric[NS],norm;
    int k[NS],time;
} STATE, *STATEPTR;

```

```

viterbi (file_name,size,pstate,pstar)

```

```

    STATEPTR pstate;
    STARPTR pstar;
    char *file_name;
    int size;

```

```

{
    SAMPLE rec0,rec1,u;
    unsigned int j,i,h,s,t,ssurv[NS];
    float branch[4],L0,L1,mmetric[NS];

```

```

    if (pstate == NULL)
    {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1) return(3);

```

```

        TIME=T-1;
        printf("\nTrunc. Length = %d",T);
        for(j=0;j<NS;j++)
        {
            K[j]=2*((j^j>>2^j>>3^j>>5)%2)+((j^j>>1^j>>2^j>>3)%2);
            METRIC[j]=10000;
            SURV[j]=0;
        }
        METRIC[0]=0;
        NORM=0;
    }

```

```

/*****

```

```

if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
if(length_input_fifo(0)%2 != 0) return(34);

```

```

while(length_input_fifo(0) > 1)
{
    if(length_output_fifo(0)==maxlength_output_fifo(0))return(0);
    get(0,&rec0);
    get(0,&rec1);
    branch[0]=(rec0+SCALE)*(rec0+SCALE)+(rec1+SCALE)*(rec1+SCALE);
    branch[1]=(rec0+SCALE)*(rec0+SCALE)+(rec1-SCALE)*(rec1-SCALE);
    branch[2]=(rec1+SCALE)*(rec1+SCALE)+(rec0-SCALE)*(rec0-SCALE);
    branch[3]=(rec0-SCALE)*(rec0-SCALE)+(rec1-SCALE)*(rec1-SCALE);
    /*****/
    h=TIME--;
    if(TIME<0) TIME += T;
    s=~(01<<h);

    for(j=0;j<NS;j++) {ssurv[j]=SURV[j]; mmetric[j]=METRIC[j]-NORM;}

    NORM=100000;
    for(j=0;j<NS;j++)
    {
        i=j>>1;
        L0=mmetric[i]+branch[K[j]];
        L1=mmetric[i|32]+branch[3-K[j]];
        if(L1 < L0) {METRIC[j]=L1; SURV[j]=(ssurv[(j>>1)|32]& s)|((j&01)<<h);}
        else      {METRIC[j]=L0; SURV[j]=(ssurv[j>>1]& s)|((j&01)<<h);}
        if( METRIC[j]<NORM) {NORM=METRIC[j]; t=j;}
    }

    u=(SAMPLE)((SURV[t] >>TIME)&01);
    put(0,u);
    }
    return (0) ;
}

```

```
{***** ALOHA.PAS *****}
```

```
PROGRAM multiple_channel_slotted_aloha (input, output);
{$I sci-graf.inc}
```

```
CONST
```

```
    channel_max           = 10;
    station_max           = 20;
    trial_max             = 20;
    file_size_max         = 100;
    default_plot_interval = 10;
    dot_display_freq      = 1000;
    post_text_file        = 'posttext.dat';
    plot_disk_file        = 'plotfile.dat';
    post_delay_file       = 'posdelay.dat';
    post_throughput_file  = 'posthrpt.dat';
    post_queue_file       = 'posqueue.dat';
    input_file            = 'aloha.tdt';
```

```
TYPE
```

```
    str80                = packed array[1..80] of char;
    label_name           = string[80];
    plot_type            = (delay_curve, throughput, queue_size);
    channel_number       = 0..channel_max;
    station_number       = 0..station_max;
    trial_number         = 0..trial_max;
    event_kind           = (arrival, transmission, re_transmission,
                           resolution, departure, END_of_slot);
    channel_state        = (idle, busy, collision);
    link = ^event;
    event =
```

```
    RECORD
```

```
        fptr, bptr : link;
        kind : event_kind;
        time : double;
        arrival_time : double;
        trial_no : trial_number;
        channel_no : channel_number;
        station_no : station_number
```

```
    END;
```

```
VAR
```

```
    num_of_arv, num_of_depart : ARRAY [station_number] OF longint;
    mean_time                 : ARRAY [event_kind] OF double;
    state                     : ARRAY [channel_number] OF chan-
nel_state;
    glma                      : ARRAY [1..55] OF double;
    average_delay, variance_delay : ARRAY [station_number] OF double;
```

```
    user, population, station, channel, uniform_seed, exp_seed : integer;
    max_traffic, input_rate, max_time, service_time             : double;
    current_event, base                                         : link;
    event_index                                                  : event_kind;
    station_total                                                : station_number;
    channel_total                                                : channel_number
```

```

backlog, glinext, glinextp                                : integer;

( graphics oriented variables )
configuration_file, text_file                             : text;
plot_file, delay_file, throughput_file, queue_file        : text;
plot_yneno, plot_int                                       : integer;
plot_channel, plot_interval                               : integer;
plot_factor                                                : double;
plot_switch                                                : boolean;
plot                                                        : plot_type;
name1, name2, name3, name4, name5, name6                  : str80;

```

```

($I header.inc)

```

```

FUNCTION ran3(VAR idum: integer): double;

```

```

CONST

```

```

    mbig      = 4.0e6;
    mseed     = 1618033.0;
    mz        = 0.0;
    fac       = 2.5e-7; (* 1/mbig *)

```

```

VAR

```

```

    i,ii,k     : integer;
    mj,mk      : double;

```

```

BEGIN

```

```

    if (idum < 0) then

```

```

        BEGIN

```

```

            mj := mseed+idum;
            if mj>=0.0 then mj := mj-mbig*trunc(mj/mbig)
                else mj := mbig-abs(mj)+mbig*trunc(abs(mj)/mbig);

```

```

            glma[55] := mj;

```

```

            mk := 1;

```

```

            for i := 1 to 54 do

```

```

                BEGIN

```

```

                    ii := 21*i mod 55;

```

```

                    glma[ii] := mk;

```

```

                    mk := mj-mk;

```

```

                    if (mk < mz) then mk := mk+mbig;

```

```

                    mj := glma[ii]

```

```

                END;

```

```

            for k := 1 to 4 do

```

```

                BEGIN

```

```

                    for i := 1 to 55 do

```

```

                        BEGIN

```

```

                            glma[i] := glma[i]-glma[1+((i+30) mod 55)];

```

```

                            if (glma[i] < mz) then glma[i] := glma[i]+mbig

```

```

                        END

```

```

                    END;

```

```

            glinext := 0;

```

```

            glinextp := 31;

```

```

            idum := 1

```

```

        END;

```

```

    glinext := glinext+1;

```

```

    if (glinext = 56) then glinext := 1;

```

```

glinextp := glinextp+1;
if (glinextp = 56) then glinextp := 1;
mj := glma[glinext]-glma[glinextp];
if (mj < mz) then mj := mj+mbig;
glma[glinext] := mj;
ran3 := mj*fac
END;

procedure str_to_pack(      in_str : label_name;
                           var pack_str : str80      );
var
  max_length, element      : integer;
BEGIN
  max_length := length(in_str);
  for element := 1 to max_length do
    pack_str[element] := in_str[element];
  pack_str[max_length+1] := chr(0);
END;

procedure plot_initialize;
BEGIN
  case plot_int of
    1 : plot := delay_curve;
    2 : plot := throughput;
    3 : plot := queue_size;
  END;

  auto_select_display;
  virtual_display(YES);
  hrange(max_time, 0.0);
  case plot of
    delay_curve :
      BEGIN
        str_to_pack('Delay vs Time', name1);
        str_to_pack('Simulation Time (slots)', name2);
        str_to_pack('Average Delay (slots)', name3);
        vrange(trial_max*1.0, 0.0);
      END;
    throughput :
      BEGIN
        str_to_pack('Throughput vs Time', name1);
        str_to_pack('Simulation Time (slots)', name2);
        str_to_pack('Average Throughput', name3);
        vrange(1.0 * channel_total, 0.0);
      END;
    queue_size :
      BEGIN
        str_to_pack('Queue Size vs Time', name1);
        str_to_pack('Simulation Time (slots)', name2);
        str_to_pack('Average Queue Size', name3);
        vrange(10.0, 0.0);
      END;
  END;

```

```

    END; { END case }
graph_type(ORDINARY);
line_connect(NO);
display_onscreen(YES);
symbols(DOT, DOT, DOT, DOT, DOT, SQUARE);
hlbl_prec(1);
vlbl_prec(3);
clear_display(NO, YES);

```

```

title(CENTER, name1);
haxis_lbl(CENTER, name2);
vaxis_lbl(CENTER, name3);

```

```

display_window(643, 0, 900, 0);
graph_init;
END;

```

```

procedure plot_graph(disk_file : label_name);
BEGIN
    str_to_pack('', name1);
    str_to_pack('', name2);
    str_to_pack('', name3);
    str_to_pack('', name4);
    str_to_pack('', name5);
    str_to_pack(disk_file, name6);
    plot_pairs(name1, name2, name3, name4, name5, name6, 1);
    clear_display(NO, NO);
    background(NO);
END;

```

```

FUNCTION expdev(VAR idum: integer): double;
BEGIN
    expdev := -ln(ran3(idum))/mean_time[arrival]
END;

```

```

PROCEDURE collision_resolution( trial: trial_number;
                                VAR next_transmission_time: double);
VAR
    K_max : double;
BEGIN
    K_max := mean_time[re_transmission];
    next_transmission_time := K_max * ran3(uniform_seed);
END;

```

```

PROCEDURE inter_arrival_time (VAR next_arrival: double);
BEGIN
    next_arrival := expdev(exp_seed);
END;

```

```

PROCEDURE generate_event (evkind      : event_kind;
                        current_time: double;
                        arv_time    : double;
                        trial       : trial_number;
                        channel     : channel_number;
                        station     : station_number);

VAR
    event, new_event : link;
    delay : double;
BEGIN
    new(new_event);
    CASE evkind OF
        arrival: { next arrival }
            BEGIN
                inter_arrival_time(delay);
                arv_time := current_time + delay;
                new_event^.time := arv_time;
            END;
        transmission: { time to next slot }
            BEGIN
                new_event^.time := trunc(current_time) + 1.0001;
            END;
        resolution: { resolution is scheduled at the middle of the slot }
            BEGIN
                new_event^.time := current_time + 0.5;
            END;
        re_transmission: { schedule next transmission time }
            BEGIN
                collision_resolution(trial, delay);
                new_event^.time := trunc(current_time + delay + 0.4999) + 0.0001;
            END;
        departure: { total packet transmission time is one slot }
            BEGIN
                new_event^.time := current_time + 0.4999;
            END;
        END_of_slot:
            BEGIN
                new_event^.time := current_time + 1.0;
            END;
    END; { END CASE }

    WITH new_event^ DO
        BEGIN
            kind := evkind;
            arrival_time := arv_time;
            trial_no := trial;
            channel_no := channel;
            station_no := station;
        END; { with }
        event := base;
        REPEAT
            event := event^.bptr
        UNTIL new_event^.time >= event^.time;
        new_event^.fptr := event^.fptr;
    
```

```

new_event^.bptr := event;
event^.fptr^.bptr := new_event;
event^.fptr := new_event;
END; { generate_event }

```

```

BEGIN

```

```

{ initialize variables }
station_total := 1;
exp_seed := -1;
uniform_seed := -1;

```

```

reset(configuration_file, input_file);
readln(configuration_file, population);
readln(configuration_file, channel_total);
FOR user := 1 TO station_max DO

```

```

    BEGIN

```

```

        readln(configuration_file, input_rate);
        mean_time[arrival] := mean_time[arrival] + input_rate;
    END;

```

```

readln(configuration_file, max_time);
readln(configuration_file, plot_int);
readln(configuration_file, plot_ynsno);
readln(configuration_file, mean_time[re_transmission]);
close(configuration_file);

```

```

max_traffic := mean_time[arrival];
plot_factor := (max_traffic * max_time)
                / (file_size_max * default_plot_interval);
plot_interval := default_plot_interval;
if plot_factor > 1.0 then
    plot_interval := default_plot_interval * trunc(plot_factor + 1.0);

```

```

rewrite(text_file, post_text_file);
rewrite(delay_file, post_delay_file);
rewrite(throughput_file, post_throughput_file);
rewrite(queue_file, post_queue_file);
rewrite(plot_file, plot_disk_file);
writeln(text_file, 'Multiple Access Protocol - ALOHA (multiple channels)');
writeln(text_file);
writeln(text_file, 'Simulation Results:');
writeln(text_file, 'Simulated Time      Average Delay      Throughput');
writeln(text_file, '=====');

```

```

plot_switch := TRUE;

```

```

case plot_ynsno of

```

```

    1 : plot_switch := TRUE;
    0 : plot_switch := FALSE;

```

```

END; { END case }

```

```

IF plot_int = 0 THEN plot_switch := FALSE;
if (not plot_switch) then writeln('Running...');

```

```

{ create an empty event ring }
new(base);

```

```

WITH base^ DO
  BEGIN
    fptr := base;
    bptr := base;
    time := 0.0
  END;

{ generate arrival at each station }
FOR station := 1 TO station_total DO
  BEGIN
    average_delay[station] := 0.0;
    variance_delay[station] := 0.0;
    num_of_arv[station] := 0;
    num_of_depart[station] := 0;
    FOR channel := 1 TO channel_total DO
      generate_event(arrival, 0.0, 0.0, 0, channel, station);
    END;

{ generate initial conditions for each channel }
FOR channel := 1 TO channel_total DO
  BEGIN
    generate_event(END_of_slot, 0.0, 0.0, 0, channel, 0);
    state[channel] := idle;
  END;

if plot_switch then plot_initialize;

REPEAT
  current_event := base^.fptr;
  WITH current_event^ DO
    CASE kind OF
      arrival:
        BEGIN
          num_of_arv[station_no] := num_of_arv[station_no] + 1;
          generate_event(arrival, time, arrival_time, trial_no, channel_no,
station_no);
          generate_event(transmission, time, arrival_time, trial_no,
channel_no, station_no);
        END;
      transmission, re_transmission:
        BEGIN
          trial_no := trial_no + 1;
          CASE state[channel_no] OF
            idle: state[channel_no] := busy;
            busy: state[channel_no] := collision;
            collision: state[channel_no] := collision;
          END; { case }
          generate_event(resolution, time, arrival_time, trial_no, chan-
nel_no, station_no);
        END;
      resolution:
        BEGIN
          IF state[channel_no] = collision
            THEN BEGIN

```

```

        generate_event(re_transmission, time, arrival_time, trial_no,
channel_no, station_no);
        END
    ELSE BEGIN
        generate_event(departure, time, arrival_time, trial_no,
channel_no, station_no);
        END;
    END;
departure:
BEGIN
    num_of_depart[station_no] := num_of_depart[station_no] + 1;
    service_time := current_event^.time - current_event^.arrival_time
    average_delay[station_no] := average_delay[station_no] + service_
time;
    variance_delay[station_no] := variance_delay[station_no]
        + service_time * service_time;
    if (num_of_depart[station_no] mod plot_interval) = 0 then
    BEGIN
        writeln(delay_file, current_event^.time:10:3,
            average_delay[station_no]/num_of_depart[station_no]:10:5);
        writeln(throughput_file, current_event^.time:10:3,
            num_of_depart[station_no]/current_event^.time:10:5);
        writeln(text_file, current_event^.time:10:3,
            average_delay[station_no]/num_of_depart[station_no]:20:5,
            num_of_depart[station_no]/current_event^.time:18:5);
        if plot_switch then
        BEGIN
            reset(plot_file, plot_disk_file);
            case plot of
                delay_curve:
                    writeln(plot_file, current_event^.time:10:3,
                        average_delay[station_no]/num_
_of_depart[station_no]:10:5);
                throughput:
                    writeln(plot_file, current_event^.time:10:3,
                        num_of_depart[station_no]/current_event^.time:10:5);
            END; { END case }
            close(plot_file);
            plot_graph(plot_disk_file);
        END;
    END;
END;
END_of_slot:
BEGIN
    state[channel_no] := idle;
    generate_event(END_of_slot, time, arrival_time, trial_no, chan-
nel_no, station_no);
    if (((trunc(current_event^.time) mod dot_display_freq) = 0)
        and (not plot_switch)) then write('.');
    END;
END;
base^.fptr := current_event^.fptr;
current_event^.fptr^.bptr := base;
backlog := current_event^.trial_no;
dispose(current_event);

```

```

UNTIL ((base^.fptr^.time >= max_time) or (backlog >= trial_max));

if plot_switch then graph_close;

close(delay_file);
close(throughput_file);
close(queue_file);

{ collect statistics }
writeln(text_file);
writeln(text_file, 'Final Statistics:');
writeln(text_file, '=====');
writeln(text_file, ' Maximum simulation slots: ', max_time:10:2);
writeln(text_file, ' Number of Channels :', channel_total:5);
writeln(text_file, ' Maximum Retransmission Delay : ',
          mean_time[re_transmission]:8:2, ' slots');
writeln(text_file, ' Average arrival rate :', mean_time[arrival]:10:5,
          ' packets per slot');
writeln(text_file, '
          Number of
(slots)');
          Delay
writeln(text_file, ' Station arrival departure average
variance');
FOR station := 1 TO station_total DO
  BEGIN
    IF num_of_depart[station] = 0
      THEN average_delay[station] := 0.0
      ELSE average_delay[station] := average_delay[station]
                                   / num_of_depart[station];
    IF num_of_depart[station] <= 1
      THEN variance_delay[station] := 0.0
      ELSE variance_delay[station] := (variance_delay[station]
- num_of_depart[station] * average_delay[station] * average_delay[sta-
tion])
                                   / (num_of_depart[station] - 1);
    writeln(text_file, station:10,
            num_of_arv[station]:10, num_of_depart[station]:10,
            average_delay[station]:15:5, variance_delay[station]:15:5);
  END;
close(text_file);
END.

```

```
{***** BIDLOOP.PAS *****}
```

```
program bi_directional_loop_network(input, output);
{$I sci-graf.inc}
```

```
const
```

```
station_max      = 8;
channel_max      = 16;
session_max      = 64;
queue_max        = 20;
file_size_max    = 100;
default_plot_interval = 10;
dot_display_freq = 1000;
post_text_file   = 'posttext.dat';
plot_disk_file   = 'plotfile.dat';
post_delay_file  = 'posdelay.dat';
post_throughput_file = 'posthrpt.dat';
post_queue_file  = 'posqueue.dat';
input_network_file = 'bidloop.tdt';
```

```
type
```

```
str80      = packed array[1..80] of char;
label_name = string[80];
plot_type  = (delay_curve, throughput, queue_size);
node_number = 0..station_max;
session_number = 0..session_max;
channel_number = 0..channel_max;
event_kind  = (arrival, rcv_pkt, departure);
link        = ^event;
event       =
    record
        fptr, bptr : link;
        kind : event_kind;
        time : double;
        arrival_time : double;
        pkt_time : double;
        node_no : node_number;
        session_no : session_number
    end;
```

```
var
```

```
origin, session, channel, node, next, exp_seed      : integer;
pkt_length, pkt_time, max_pkt_time, max_time, service_time : double;
dummy, fixed_pkt_length                             : double;
current_event, base                                  : link;
```

```
{ graphics oriented variables }
```

```
network_file, text_file      : text;
plot_file, delay_file, throughput_file, queue_file      : text;
plot_yesno, plot_int, plot_session      : integer;
plot_channel, plot_interval      : integer;
plot_session_in, plot_session_out, plot_channel_in      : integer;
plot_channel_out                : integer;
plot_factor                     : double;
plot_switch                     : boolean;
```

```

plot                                     : plot_type;
name1, name2, name3, name4, name5, name6 : str80;

```

```

{ session oriented variables }
session_total           : session_number;
max_traffic             : double;
num_of_arv, num_of_depart : array [session_number] of longint;
mean_pkt_length, traffic_rqmt : array[session_number] of double;
session_rqmt            : array[session_number,1..2] of node_number
average_delay, variance_delay : array [session_number] of double;
last_departure_time      : array [session_number] of double;
inter_departure_time      : array [session_number] of double;

```

```

{ channel oriented variables }
channel_total           : channel_number;
back_log               : array[channel_number] of integer;
last_arrival_time, wait_time, capacity : array[channel_number] of double;
route                  : array[node_number,session_number] of channel_number

```

```

{ node oriented variables }
origin_no, destination_no, node_total : node_number;
half                                  : node_number;
node_process_time                    : array[node_number] of double;
link_matrix                          : array[channel_number,1..2] of node_number;
short_path                          : array[node_number,node_number] of channel_number;
row, column                          : node_number;

```

```

{ variables for random number generation }
glinext, glinextp           : integer;
glma                        : array [1..55] of double;

```

```

{
the network is characterized by three matrices:

```

1. link_matrix

	from	to	capacity
channel_number	node_no	node_no	bits/sec

2. session_matrix

	origin	destination	traffic_rqmt	mean_pkt_length
session_number	node_no	node_no	packets/sec	bits/packet

3. rotuing_matrix

session_number

node_process_time

node_number

out_bound_queue channel_number

seconds

}

{ \$I header.inc }

function ran3(var idum: integer): double;

const

mbig = 4.0e6;

mseed = 1618033.0;

mz = 0.0;

fac = 2.5e-7; (* 1/mbig *)

var

i,ii,k : integer;

mj,mk : double;

begin

if (idum < 0) then

begin

mj := mseed+idum;

if mj>=0.0 then mj := mj-mbig*trunc(mj/mbig)

else mj := mbig-abs(mj)+mbig*trunc(abs(mj)/mbig);

glma[55] := mj;

mk := 1;

for i := 1 to 54 do

begin

ii := 21*i mod 55;

glma[ii] := mk;

mk := mj-mk;

if (mk < mz) then mk := mk+mbig;

mj := glma[ii]

end;

for k := 1 to 4 do

begin

for i := 1 to 55 do

begin

glma[i] := glma[i]-glma[1+((i+30) mod 55)];

if (glma[i] < mz) then glma[i] := glma[i]+mbig

end

end;

glinext := 0;

glinextp := 31;

idum := 1

end;

glinext := glinext+1;

```

if (glinext = 56) then glinext := 1;
glinextp := glinextp+1;
if (glinextp = 56) then glinextp := 1;
mj := glma[glinext]-glma[glinextp];
if (mj < mz) then mj := mj+mbig;
glma[glinext] := mj;
ran3 := mj*fac
end;

```

```

procedure str_to_pack(      in_str : label_name;
                           var pack_str : str80      );

```

```

var
  max_length, element      : integer;

begin
  max_length := length(in_str);
  for element := 1 to max_length do
    pack_str[element] := in_str[element];
  pack_str[max_length+1] := chr(0);
end;

```

```

procedure plot_initialize;
begin

```

```

  case plot_int of
    1 : plot := delay_curve;
    2 : plot := throughput;
    3 : plot := queue_size;
  end;

```

```

  auto_select_display;
  virtual_display(YES);
  hrange(max_time, 0.0);
  case plot of

```

```

    delay_curve :

```

```

      begin

```

```

        str_to_pack('Delay vs Time', name1);
        str_to_pack('Simulation Time (seconds)', name2);
        str_to_pack('Average Delay (seconds)', name3);
        vrange(queue_max * max_pkt_time, 0.0);

```

```

      end;

```

```

    throughput :

```

```

      begin

```

```

        str_to_pack('Throughput vs Time', name1);
        str_to_pack('Simulation Time (seconds)', name2);
        str_to_pack('Average Throughput', name3);
        vrange(1.0, 0.0);

```

```

      end;

```

```

    queue_size :

```

```

      begin

```

```

        str_to_pack('Queue Size vs Time', name1);
        str_to_pack('Simulation Time (seconds)', name2);
        str_to_pack('Average Queue Size', name3);
        vrange(queue_max, 0.0);

```

```

        end;
    end; { end case }
graph_type(ORDINARY);
line_connect(NO);
display_onscreen(YES);
symbols(DOT, DOT, DOT, DOT, DOT, SQUARE);
hlbl_prec(1);
vlbl_prec(3);
clear_display(NO, YES);

title(CENTER, name1);
haxis_lbl(CENTER, name2);
vaxis_lbl(CENTER, name3);

display_window(643, 0, 900, 0);
graph_init;
end;

```

```

procedure plot_graph(disk_file : label_name);
begin
    str_to_pack('', name1);
    str_to_pack('', name2);
    str_to_pack('', name3);
    str_to_pack('', name4);
    str_to_pack('', name5);
    str_to_pack(disk_file, name6);
    plot_pairs(name1, name2, name3, name4, name5, name6, 1);
    clear_display(NO, NO);
    background(NO);
end;

```

```

function expdev(var idum: integer; session: integer): double;
begin
    expdev := -ln( ran3(idum) )/traffic_rqmt[session];
end;

```

```

procedure inter_arrival_time (var next_arrival : double;
                             session : integer);
begin
    next_arrival := expdev(exp_seed, session);
end;

```

```

procedure pkt_distribution(var pkt_size: double; session: integer);
begin
    pkt_size := mean_pkt_length[session];
end;

```

```

procedure generate_event (evkind      : event_kind;

```

```

                                current_time: double;
                                arv_time    : double;
                                pkt_len     : double;
                                node        : node_number;
                                session     : session_number);

var
    event,new_event : link;
    out_bound_queue : channel_number;
    delay : double;

begin
    new(new_event);
    case evkind of
        arrival: { next arrival }
            begin
                inter_arrival_time(delay, session);
                arv_time := current_time + delay;
                new_event^.time := arv_time;
            end;
        departure:
            begin
                new_event^.time := current_time + node_process_time[node];
            end;
        rcv_pkt:
            begin
                out_bound_queue := route[node,session];
                new_event^.time := current_time + node_process_time[node]
                    + pkt_len + wait_time[out_bound_queue];
            end;
    end; { end case }

    with new_event^ do
        begin
            kind := evkind;
            arrival_time := arv_time;
            pkt_time := pkt_len;
            node_no := node;
            session_no := session;
        end; { with }
    event := base;
    repeat
        event := event^.bptr
        until new_event^.time >= event^.time;
    new_event^.fptr := event^.fptr;
    new_event^.bptr := event;
    event^.fptr^.bptr := new_event;
    event^.fptr := new_event;
end; { generate_event }

begin
    { read in network size }
    reset(network_file, input_network_file);
    readln(network_file, node_total);

```

```

{ read in plotting switch }
readln(network_file, plot_session_in);
readln(network_file, plot_session_out);
readln(network_file, plot_channel_in);
readln(network_file, plot_channel_out);
plot_session := 1;
plot_channel := 1;
plot := delay_curve;
plot_switch := TRUE;

{ read in netowrk matrices }
session := 1;
for row := 1 to station_max do { build session matrix from traffic matrix
  for column := 1 to station_max do begin
    readln(network_file, traffic_rqmt[session]);
    if (( abs(traffic_rqmt[session]) > 1.0e-6) and
        ( row <= node_total ) and ( column <= node_total ) ) then begin
      session_rqmt[session,1] := row;
      session_rqmt[session,2] := column;
      if ((plot_session_in = row) and (plot_session_out = column))
        then plot_session := session;
      mean_pkt_length[session] := 1.0;
      session := session + 1;
    end;
  end;
session_total := session - 1;

channel_total := 2 * node_total;
for channel := 1 to node_total do begin { build link matrix }
  readln(network_file, capacity[channel]);
  link_matrix[channel,1] := channel;
  link_matrix[channel,2] := (channel mod node_total) + 1;
  if ((plot_channel_in = link_matrix[channel,1]) and
      (plot_channel_out = link_matrix[channel,2]))
    then plot_channel := channel;
end;

{ read in dummy entries }
if (node_total < station_max) then
  for channel := (node_total + 1) to (channel_max div 2) do
    readln(network_file, dummy);

{ continue to build link matrix }
for channel := 1 to node_total do begin
  readln(network_file, capacity[channel+node_total]);
  link_matrix[channel+node_total,1] := (channel mod node_total) + 1;
  link_matrix[channel+node_total,2] := channel;
  if ((plot_channel_in = link_matrix[channel+node_total,1]) and
      (plot_channel_out = link_matrix[channel+node_total,2]))
    then plot_channel := channel+node_total;
end;

{ read in dummy entries }
if (node_total < station_max) then
  for channel := (node_total + 1) to (channel_max div 2) do

```

```

    readln(network_file, dummy);

    { build shortest path matrix }
    half := node_total div 2;
    for origin := 1 to node_total do
        for next := 1 to node_total do
            begin
                if (((node_total + origin - next) mod node_total) > half)
                then short_path[origin, next] := origin
                else begin
                    short_path[origin, next] := origin + (node_total-1);
                    if (short_path[origin, next] <= node_total) then
                        short_path[origin, next] := short_path[origin, next] + node_to-
tal;
                end;
            end;
        end;

    for node := 1 to node_total do { build routing matrix for biloop only }
        begin
            node_process_time[node] := 0.0;
            for session := 1 to session_total do
                route[node, session] := short_path[node, session_rqmt[session, 2]];
            end;

        readln(network_file, max_time);
        readln(network_file, plot_int);
        readln(network_file, plot_yesno);

        close(network_file);

    { echo the three matrices }
    rewrite(text_file, post_text_file);
    writeln(text_file, 'Store-and-Forward Network - Bi-Directional Loop');
    writeln(text_file);
    writeln(text_file, 'Input');
    writeln(text_file, '====');
    writeln(text_file, ' Routing Matrix :');
    for node := 1 to node_total do { routing matrix }
        begin
            for session := 1 to session_total do
                write(text_file, route[node, session]:3);
                writeln(text_file, node_process_time[node]:10:5);
            end;
        writeln(text_file);
        writeln(text_file, ' Session Matrix :');
        for session := 1 to session_total do { session matrix }
            writeln(text_file, session_rqmt[session, 1]:3, session_rqmt[session, 2]:3,
                traffic_rqmt[session]:10:5, mean_pkt_length[session]:10:5);
        writeln(text_file);
        writeln(text_file, ' Link Matrix :');
        for channel := 1 to channel_total do { link matrix }
            writeln(text_file, link_matrix[channel, 1]:3, link_matrix[channel, 2]:3,
                capacity[channel]:10:5);
        writeln(text_file);
        writeln(text_file, 'Simulation Results of Session', plot_session:4);

```

```
writeln(text_file, 'Simulated Time      Average Delay      Throughput');
writeln(text_file, '=====');
```

```
{ initialize variables }
exp_seed := -1;
for channel := 1 to channel_total do
begin
    wait_time[channel] := 0.0;
    last_arrival_time[channel] := 0.0;
    back_log[channel] := 0;
end;
max_traffic := 0.0;
for session := 1 to session_total do
begin
    last_departure_time[session] := 0.0;
    average_delay[session] := 0.0;
    variance_delay[session] := 0.0;
    num_of_arv[session] := 0;
    num_of_depart[session] := 0;
    if max_traffic <= traffic_rqmt[session] then
        max_traffic := traffic_rqmt[session];
    end;
plot_factor := (max_traffic * max_time)
                / (file_size_max * default_plot_interval);
plot_interval := default_plot_interval;
if plot_factor > 1.0 then
    plot_interval := default_plot_interval * trunc(plot_factor + 1.0);

rewrite(delay_file, post_delay_file);
rewrite(throughput_file, post_throughput_file);
rewrite(queue_file, post_queue_file);
rewrite(plot_file, plot_disk_file);

case plot_yesno of
    1 : plot_switch := TRUE;
    0 : plot_switch := FALSE;
end; { end case }
IF plot_int = 0 THEN plot_switch := FALSE;
if (not plot_switch) then writeln('Running...');

{ create an empty event ring }
new(base);
with base^ do
begin
    fptr := base;
    bptr := base;
    time := 0.0
end;

{ generate arrival for each session }
max_pkt_time := 0.0;
for session := 1 to session_total do
begin
    origin := session_rqmt[session,1];
    channel := route[origin,session];
```

```

    pkt_distribution(pkt_length, session);
    pkt_time := pkt_length / capacity[channel];
    if max_pkt_time <= pkt_time then max_pkt_time := pkt_time;
    generate_event(arrival, 0.0, 0.0, pkt_time, origin, session);
end;

if plot_switch then plot_initialize;

repeat
    current_event := base^.fptr;
    with current_event^ do
        case kind of
            arrival:
                begin
                    num_of_arv[session_no] := num_of_arv[session_no] + 1;
                    generate_event(arrival, time, arrival_time, pkt_time, node_no,
session_no);
                    channel := route[node_no, session_no];
                    if back_log[channel] = 0 then
                        begin
                            wait_time[channel] := 0.0;
                            last_arrival_time[channel] := arrival_time;
                            back_log[channel] := back_log[channel] + 1;
                        end
                    else
                        begin
                            wait_time[channel] := wait_time[channel] + pkt_time
                                - ( arrival_time - last_arrival_time[channel] );
                            last_arrival_time[channel] := arrival_time;
                            if wait_time[channel] <= 0.0 then
                                wait_time[channel] := 0.0;
                            back_log[channel] := back_log[channel] + 1;
                        end;
                    generate_event(rcv_pkt, time, arrival_time, pkt_time, node_no,
session_no);
                    if ((num_of_arv[plot_session] mod dot_display_freq = 0)
                        and (not plot_switch)) then write(' ');
                    end;
                departure:
                    begin
                        num_of_depart[session_no] := num_of_depart[session_no] + 1;
                        service_time := current_event^.time - current_event^.arrival_time
                        average_delay[session_no] := average_delay[session_no] + service_
time;
                        variance_delay[session_no] := variance_delay[session_no]
                            + service_time * service_time;
                        inter_departure_time[session_no] := current_event^.time
                            - last_departure_time[session_no];
                        last_departure_time[session_no] := current_event^.time;

                        if (num_of_depart[session_no] mod plot_interval) = 0 then
                            begin
                                if plot_session = session_no then
                                    begin

```

```

        writeln(delay_file, current_event^.time:10:3,
        average_delay[session_no]/num_of_depart[session_no]:10:5)
        writeln(throughput_file, current_event^.time:10:3,
        num_of_depart[session_no]/num_of_arv[session_no]:10:5);
        writeln(text_file, current_event^.time:10:3,
        average_delay[session_no]/num_of_depart[session_no]:20:5,
        num_of_depart[session_no]/num_of_arv[session_no]:18:5);
    end;
    if plot_switch then
    begin
        reset(plot_file, plot_disk_file);
        case plot of
            delay_curve: if plot_session = session_no then
                writeln(plot_file, current_event^.time:10:3,
                average_delay[session_no]/num_of_depart[session_no]:10:5);
            throughput: if plot_session = session_no then
                writeln(plot_file, current_event^.time:10:3,
                num_of_depart[session_no]/num_of_arv[session_no]:10:5);
        end; { end case }
        close(plot_file);
        plot_graph(plot_disk_file);
    end;
end;
end;

rcv_pkt:
begin
    channel := route[node_no, session_no];
    back_log[channel] := back_log[channel] - 1;

    if (num_of_depart[session_no] mod plot_interval) = 0 then
    begin
        if plot_channel = channel then
            writeln(queue_file, current_event^.time:10:5,
            back_log[channel]:5);

            if plot_switch then
            begin
                reset(plot_file, plot_disk_file);
                if plot = queue_size then
                    if plot_channel = channel then
                        writeln(plot_file, current_event^.time:10:5,
                        back_log[channel]:5);
                    close(plot_file);
                    plot_graph(plot_disk_file);
                end;
            end;

            node := link_matrix[channel, 2];
            if node = session_rqmt[session_no, 2]
            then
                { final destination }
                begin

```

```

        generate_event(departure, time, arrival_time, pkt_time, node,
session_no)
    end
else
    { route to next node }
    begin
        channel := route[node,session_no];
        if back_log[channel] = 0 then
            begin
                wait_time[channel] := 0.0;
                back_log[channel] := back_log[channel] + 1;
                last_arrival_time[channel] := time;
            end
        else
            begin
                wait_time[channel] := wait_time[channel] + pkt_time
                    - ( time - last_arrival_time[channel] );
                last_arrival_time[channel] := time;
                if wait_time[channel] <= 0.0 then
                    wait_time[channel] := 0.0;
                back_log[channel] := back_log[channel] + 1;
            end;
            generate_event(rcv_pkt, time, arrival_time, pkt_time, node,
session_no);
        end;
    end;
end;
base^.fptr := current_event^.fptr;
current_event^.fptr^.bptr := base;
dispose(current_event);
until ((base^.fptr^.time >= max_time)
    or (back_log[plot_channel] >= queue_max));

if plot_switch then graph_close;

close(delay_file);
close(throughput_file);
close(queue_file);

{ collect statistics }
writeln(text_file);
writeln(text_file, 'Final Statistics:');
writeln(text_file, '=====');
writeln(text_file, ' maximum simulated time: ', max_time:10:2, ' seconds');
for session := 1 to session_total do
    writeln(text_file, ' average arrival rate of session', session:5, ' is',
        traffic_rqmt[session]:10:5, ' packets per second');
    writeln(text_file, '
                                number of
                                delay
(seconds)');
    writeln(text_file, ' session arrival departure average
variance');
    for session := 1 to session_total do
        begin
            if num_of_depart[session] = 0
                then average_delay[session] := 0.0

```

```

        else average_delay[session] := average_delay[session]
                                     / num_of_depart[session];
    if num_of_depart[session] <= 1
        then variance_delay[session] := 0.0
        else variance_delay[session] := (variance_delay[session]
- num_of_depart[session] * average_delay[session] * average_delay[ses-
sion])
                                     / (num_of_depart[session] - 1);
    writeln(text_file, session:10,
            num_of_arv[session]:10, num_of_depart[session]:10,
            average_delay[session]:15:5, variance_delay[session]:15:5);
end;
close(text_file);
end.

```

```

(***** CSMA.PAS *****/

```

```

PROGRAM carrier_sense_multiple_access (input, output);
($I sci-graf.inc)

```

```

CONST

```

```

    persistency           = 0.0;
    exit_count            = 20;
    channel_max            = 10;
    station_max           = 20;
    file_size_max         = 100;
    default_plot_interval = 10;
    dot_display_freq      = 1000;
    post_text_file        = 'posttext.dat';
    plot_disk_file        = 'plotfile.dat';
    post_delay_file       = 'posdelay.dat';
    post_throughput_file  = 'posthrpt.dat';
    post_queue_file       = 'posqueue.dat';
    input_file            = 'csma.tdt';

```

```

TYPE

```

```

    str80                 = packed array[1..80] of char;
    label_name            = string[80];
    plot_type             = (delay_curve, throughput, queue_size);
    channel_number        = 0..channel_max;
    station_number        = 0..station_max;
    event_kind            = (arrival, transmission, re_arrival, p_sensing,
                             resolution, departure, end_of_slot);
    channel_state         = (idle, busy, collision);
    link                  = ^event;
    event                 =

```

```

        RECORD

```

```

            fptr, bptr : link;
            kind : event_kind;
            time : double;
            arrival_time : double;
            channel_no : channel_number;
            station_no : station_number

```

```

        END;

```

```

VAR

```

```

    user, population, packet_size, busy_count, backlog      : integer;
    station, channel, uniform_seed, exp_seed                : integer;
    max_traffic, max_time, service_time, offer_rate,
    count_rate, prev_depart_time, interdep_time,
    average_interdep, variance_interdep, prev_arrive_time,
    interarv_time, average_interarv,
    variance_interarv, input_rate, actual_time              : double;
    current_event, base                                     : link;
    event_index                                              : event_kind;
    station_total                                            : station_number;
    channel_total                                            : channel_number;
    glinext, glinextp                                       : integer;
    glma                                                     : ARRAY [1..55] OF double;

```

```

prev_state, curr_state           : ARRAY [channel_number] OF chan-
nel_state;
num_of_arv, num_of_fresh, num_of_depart : ARRAY [station_number] OF longint
mean_time                         : ARRAY [event_kind] OF double;
average_delay, variance_delay,
throughputs, variation_coeff      : ARRAY [station_number] OF double;

{ graphics oriented VARIables }
configuration_file, text_file    : text;
plot_file, delay_file, throughput_file, queue_file : text;
plot_ynsno, plot_int             : integer;
plot_channel, plot_interval      : integer;
plot_factor                      : double;
plot_switch                      : boolean;
plot                            : plot_type;
name1, name2, name3, name4, name5, name6 : str80;

{$I header.inc}

FUNCTION ran3(VAR idum: integer): double;
CONST
  mbig      = 4.0e6;
  mseed     = 1618033.0;
  mz        = 0.0;
  fac       = 2.5e-7; (* 1/mbig *)
VAR
  i, ii, k  : integer;
  mj, mk    : double;

BEGIN
  IF (idum < 0) THEN
    BEGIN
      mj := mseed+idum;
      IF mj>=0.0 THEN mj := mj-mbig*trunc(mj/mbig)
        else mj := mbig-abs(mj)+mbig*trunc(abs(mj)/mbig);
      glma[55] := mj;
      mk := 1;
      for i := 1 to 54 do
        BEGIN
          ii := 21*i mod 55;
          glma[ii] := mk;
          mk := mj-mk;
          IF (mk < mz) THEN mk := mk+mbig;
          mj := glma[ii]
        END;
      for k := 1 to 4 do
        BEGIN
          for i := 1 to 55 do
            BEGIN
              glma[i] := glma[i]-glma[1+((i+30) mod 55)];
              IF (glma[i] < mz) THEN glma[i] := glma[i]+mbig
            END
          END;
        END;
      glinext := 0;
      glinextp := 31;
    END;
  END;

```

```

        idum := 1
    END;
    glinext := glinext+1;
    IF (glinext = 56) THEN glinext := 1;
    glinextp := glinextp+1;
    IF (glinextp = 56) THEN glinextp := 1;
    mj := glma[glinext]-glma[glinextp];
    IF (mj < mz) THEN mj := mj+mbig;
    glma[glinext] := mj;
    ran3 := mj*fac
END;

PROCEDURE str_to_pack(      in_str : label_name;
                          VAR pack_str : str80      );
VAR
    max_length, element    : integer;
BEGIN
    max_length := length(in_str);
    for element := 1 to max_length do
        pack_str[element] := in_str[element];
    pack_str[max_length+1] := chr(0);
END;

PROCEDURE plot_initialize;
BEGIN
    CASE plot_int of
        1 : plot := delay_curve;
        2 : plot := throughput;
        3 : plot := queue_size;
    END;

    auto_select_display;
    virtual_display(YES);
    hrange(max_time/packet_size, 0.0);
    CASE plot of
        delay_curve :
            BEGIN
                str_to_pack('Delay vs Time', name1);
                str_to_pack('Simulation Time (slots)', name2);
                str_to_pack('Average Delay (slots)', name3);
                vrange(exit_count*1.0, 0.0);
            END;
        throughput :
            BEGIN
                str_to_pack('Throughput vs Time', name1);
                str_to_pack('Simulation Time (slots)', name2);
                str_to_pack('Average Throughput', name3);
                vrange(1.0, 0.0);
            END;
        queue_size :
            BEGIN
                str_to_pack('Queue Size vs Time', name1);

```

```

        str_to_pack('Simulation Time (slots)',name2);
        str_to_pack('Average Queue Size',name3);
        vrange(exit_count*1.0, 0.0);
    END;
END; { END CASE }
graph_type(ORDINARY);
line_connect(NO);
display_onscreen(YES);
symbols(DOT, DOT, DOT, DOT, DOT, SQUARE);
hlbl_prec(1);
vlbl_prec(3);
clear_display(NO, YES);

title(CENTER, name1);
haxis_lbl(CENTER, name2);
vaxis_lbl(CENTER, name3);

display_window(643, 0, 900, 0);
graph_init;
END;

PROCEDURE plot_graph(disk_file : label_name);
BEGIN
    str_to_pack('',name1);
    str_to_pack('',name2);
    str_to_pack('',name3);
    str_to_pack('',name4);
    str_to_pack('',name5);
    str_to_pack(disk_file,name6);
    plot_pairs(name1, name2, name3, name4, name5, name6, 1);
    clear_display(NO, NO);
    background(NO);
END;

FUNCTION expdev(VAR idum: integer): double;
BEGIN
    expdev := -ln(ran3(idum))/mean_time[arrival]
END;

PROCEDURE collision_resolution(VAR next_transmission_time: double);
BEGIN
    next_transmission_time := mean_time[re_arrival] * ran3(unIFORM_seed);
END;

PROCEDURE inter_arrival_time (VAR next_arrival: double);
BEGIN
    next_arrival := expdev(exp_seed);
END;

```

```

PROCEDURE generate_event (evkind      : event_kind;
                        current_time: double;
                        arv_time    : double;
                        channel      : channel_number;
                        station      : station_number);

VAR
    event,new_event : link;
    delay : double;
BEGIN
    new(new_event);
    CASE evkind OF
        arrival: { next arrival }
            BEGIN
                inter_arrival_time(delay);
                arv_time := current_time + delay;
                new_event^.time := arv_time;
            END;
        re_arrival: { schedule retransmission time }
            BEGIN
                collision_resolution(delay);
                new_event^.time := current_time + delay;
            END;
        transmission: { time to next edge+ }
            BEGIN
                new_event^.time := trunc(current_time) + 1.1;
            END;
        resolution: { resolution is scheduled within the last slot }
            BEGIN
                new_event^.time := current_time + (packet_size - 0.5);
            END;
        p_sensing : { persist on sensing at next edge }
            BEGIN
                new_event^.time := current_time + 1.0;
            END;
        departure: { total packet transmission time is one time unit }
            BEGIN
                new_event^.time := current_time + 0.4; {depart at edge}
            END;
        end_of_slot: { update channel state at edge- }
            BEGIN
                new_event^.time := trunc(current_time + 1.5) - 0.1;
            END;
    END; { END CASE }

    WITH new_event^ DO
        BEGIN
            kind := evkind;
            arrival_time := arv_time;
            channel_no := channel;
            station_no := station;
        END; { with }
    event := base;
    REPEAT
        event := event^.bptr

```

```

UNTIL new_event^.time >= event^.time;
new_event^.fptr := event^.fptr;
new_event^.bptr := event;
event^.fptr^.bptr := new_event;
event^.fptr := new_event;
END; { generate_event }

BEGIN (* main program *)

    plot := delay_curve;
    plot_switch := TRUE;

    reset(configuration_file, input_file);
    readln(configuration_file, population);
    readln(configuration_file, channel_total);
    channel_total := 1;
    FOR user := 1 TO station_max DO
        BEGIN
            readln(configuration_file, input_rate);
            mean_time[arrival] := mean_time[arrival] + input_rate;
        END;
    readln(configuration_file, max_time);
    readln(configuration_file, plot_int);
    readln(configuration_file, plot_yesno);
    readln(configuration_file, mean_time[re_arrival]);
    readln(configuration_file, packet_size);
    close(configuration_file);

    mean_time[re_arrival] := mean_time[re_arrival] * packet_size;
    mean_time[arrival] := mean_time[arrival] / packet_size;

{ initialize VARIABLES }
exp_seed := -1;
uniform_seed := -1;

station_total := 1;
backlog := 0;
max_time := max_time * packet_size;

max_traffic := mean_time[arrival];
plot_factor := (max_traffic * max_time)
                / (file_size_max * default_plot_interval);
plot_interval := default_plot_interval;
IF plot_factor > 1.0 THEN
    plot_interval := default_plot_interval * trunc(plot_factor + 1.0);

rewrite(delay_file, post_delay_file);
rewrite(throughput_file, post_throughput_file);
rewrite(queue_file, post_queue_file);
rewrite(plot_file, plot_disk_file);
rewrite(text_file, post_text_file);
writeln(text_file, 'Multiple Access Protocol - CSMA (single channel)');
writeln(text_file);
writeln(text_file, 'Simulation Results:');

```

```

writeln(text_file, 'Simulated Time      Average Delay      Throughput');
writeln(text_file, '=====');

CASE plot_yesno of
  1 : plot_switch := TRUE;
  0 : plot_switch := FALSE;
END; { END CASE }
if plot_int = 0 THEN plot_switch := FALSE;
if (not plot_switch) then writeln('Running...');

{ create an empty event ring }
new(base);
WITH base^ DO
  BEGIN
    fptr := base;
    bptr := base;
    time := 0.0
  END;

{ generate arrival at each station }
prev_depart_time := 0.0;
interdep_time := 0.0;
average_interdep := 0.0;
variance_interdep := 0.0;

prev_arrive_time := 0.0;
interarv_time := 0.0;
average_interarv := 0.0;
variance_interarv := 0.0;

FOR station := 1 TO station_total DO
  BEGIN
    average_delay[station] := 0.0;
    variance_delay[station] := 0.0;
    num_of_arv[station] := 0;
    num_of_depart[station] := 0;
    num_of_fresh[station] := 0;
    FOR channel := 1 TO channel_total DO
      generate_event(arrival, 0.0, 0.0, channel, station);
    END;

{ generate initial conditions for each channel }
FOR channel := 1 TO channel_total DO
  BEGIN
    generate_event(end_of_slot, -0.1, 0.0, channel, 0);
    prev_state[channel] := idle;
    curr_state[channel] := idle;
    busy_count := 0;
  END;

IF plot_switch THEN plot_initialize;

REPEAT
  current_event := base^.fptr;
  WITH current_event^ DO

```

```

CASE kind OF

  arrival, re_arrival:
    BEGIN
      num_of_arv[station_no] := num_of_arv[station_no] + 1;
      generate_event(transmission, time, arrival_time,
                    channel_no, station_no);
      IF current_event^.kind = arrival
      THEN BEGIN
        num_of_fresh[station_no] := num_of_fresh[station_no] + 1;
        generate_event(arrival, time, arrival_time,
                      channel_no, station_no);

        END;
        interarv_time := (current_event^.time - prev_arrive_time)
                        / packet_size;
        average_interarv := average_interarv + interarv_time;
        variance_interarv := variance_interarv + inter-
arv_time*interarv_time;
        prev_arrive_time := current_event^.time;
      END; (arrival, re_arrival)

    transmission:
      BEGIN
        CASE prev_state[channel_no] OF
          idle: BEGIN
            CASE curr_state[channel_no] OF
              idle: BEGIN
                curr_state[channel_no] := busy;
                busy_count := packet_size;
              END;
              busy, collision:
                curr_state[channel_no] := collision;
            END; ( CASE )
            generate_event(resolution, time, arrival_time,
                          channel_no, station_no);

          END; ( idle CASE )
          busy, collision:
            IF ( ran3(unIFORM_seed) < persistency ) ( persist )

            THEN generate_event(p_sensing, time, arrival_time,
                              channel_no, station_no)

            ELSE generate_event(re_arrival, time, arrival_time,
                              channel_no, station_no);

          END; (case)
        END; (transmission)

    p_sensing:
      BEGIN
        CASE prev_state[channel_no] OF
          idle: BEGIN
            CASE curr_state[channel_no] OF
              idle: BEGIN
                curr_state[channel_no] := busy;
                busy_count := packet_size;
              END;
            END;
          END;
        END;
      END;

```

```

        busy, collision:
            curr_state[channel_no] := collision;
        END; { CASE }
        generate_event(resolution, time, arrival_time,
                        channel_no, station_no);
    END; { idle CASE }
    busy, collision:
        generate_event(p_sensing, time, arrival_time,
                        channel_no, station_no);
    END; { CASE }
END; {p-sensing}

resolution:
    BEGIN
        IF curr_state[channel_no] = collision
        THEN generate_event(re_arrival, time + 0.4, arrival_time,
                            channel_no, station_no)
        ELSE generate_event(departure, time, arrival_time,
                            channel_no, station_no);
        END; {resolution}

departure:
    BEGIN
        num_of_depart[station_no] := num_of_depart[station_no] + 1;
        service_time := (current_event^.time - current_event^.ar-
rival_time)
                                / packet_size;
        average_delay[station_no] := average_delay[station_no] + service_
time;
        variance_delay[station_no] := variance_delay[station_no]
                                + service_time * service_time;

        IF (num_of_depart[station_no] mod plot_interval) = 0 THEN
            BEGIN
                actual_time := current_event^.time / packet_size;
                writeln(delay_file, actual_time:10:3,
                        average_delay[station_no]/num_of_depart[station_no]:10:5)
                writeln(throughput_file, actual_time:10:3,
                        num_of_depart[station_no]/actual_time:10:5);
                writeln(text_file, actual_time:10:3,
                        average_delay[station_no]/num_of_depart[station_no]:20:5,
                        num_of_depart[station_no]/actual_time:18:5);
                IF plot_switch THEN
                    BEGIN
                        reset(plot_file, plot_disk_file);
                        CASE plot of
                            delay_curve:
                                writeln(plot_file, actual_time:10:3,
                                        average_delay[station_no]/num-
_of_depart[station_no]:10:5);
                                throughput:
                                    writeln(plot_file, actual_time:10:3,
                                            num_of_depart[station_no]/actual_time:10:5);
                                END; { END CASE }
                    END;
            END;
        END;
    
```

```

        close(plot_file);
        plot_graph(plot_disk_file);
    END;
END;
interdep_time := (current_event^.time - prev_depart_time)
                  / packet_size;
average_interdep := average_interdep + interdep_time;
variance_interdep := variance_interdep + interdep_time*inter-
dep_time;
prev_depart_time := current_event^.time;
offer_rate := num_of_arv[station_no]/average_interarv;
count_rate := num_of_arv[station_no] * packet_size
              / current_event^.time;
FOR station := 1 TO station_total DO
    BEGIN
        throughputs[station] := num_of_depart[station]/average_interdep
        IF num_of_depart[station] <= 1
    THEN variation_coeff[station] := 1.0
        ELSE variation_coeff[station] := throughputs[station] * through
puts[station]
            * (variance_interdep - average_interdep*average_interdep
              /num_of_depart[station]) / (num_of_depart[station] - 1);

        backlog := num_of_fresh[station] - num_of_depart[station];
    END;
END; {departure}

end_of_slot:
    BEGIN
        prev_state[channel_no] := curr_state[channel_no];
        CASE curr_state[channel_no] OF
            idle : busy_count := 0;
            busy, collision :
                BEGIN
                    IF busy_count>0 THEN
                        busy_count := busy_count - 1;
                        IF (busy_count = 0) THEN
                            curr_state[channel_no] := idle;
                        END;
                    END; { CASE }
                generate_event(end_of_slot, time, arrival_time,
                              channel_no, station_no);
                if (((trunc(current_event^.time) mod (dot_display_freq
                  * packet_size)) = 0) and (not plot_switch)) then write('.');
            END; {end_of_slot}

        END; {CASE kind OF}

base^.fptr := current_event^.fptr; (* remove executed event *)
current_event^.fptr^.bptr := base; (* from ring *)
dispose(current_event); (* from memory *)
UNTIL (base^.fptr^.time >= max_time) OR (backlog >= exit_count);

IF plot_switch THEN graph_close;

```

```

close(delay_file);
close(throughput_file);
close(queue_file);

( collect statistics )
writeln(text_file);
writeln(text_file, 'Final Statistics:');
writeln(text_file, '=====');
writeln(text_file, ' Maximum simulation slots: ', max_time/packet_size:10:2)
writeln(text_file, ' Number of Channels :', channel_total:5);
writeln(text_file, ' Maximum Retransmission Delay :',
            mean_time[re_arrival]/packet_size:8:2, ' slots');
writeln(text_file, ' Propagation Ratio : ', packet_size:10);
writeln(text_file, ' Average arrival rate :',
            mean_time[arrival]*packet_size:10:5, ' packets per slot');
writeln(text_file, '
            Number of
(slots)');
            Delay
writeln(text_file, ' Station arrival departure average
variance');
FOR station := 1 TO station_total DO
    BEGIN
        IF num_of_depart[station] = 0
        THEN average_delay[station] := 0.0
        ELSE average_delay[station] := average_delay[station]
            / num_of_depart[station];

        IF num_of_depart[station] <= 1
        THEN variance_delay[station] := 0.0
        ELSE variance_delay[station] :=
            (variance_delay[station] - num_of_depart[station]
             * average_delay[station] * average_delay[station]
             / (num_of_depart[station] - 1);

        writeln(text_file, station:10,
            num_of_fresh[station]:10, num_of_depart[station]:10,
            average_delay[station]:15:5, variance_delay[station]:15:5);

    END;
close(text_file);
END.

```

```
/****** PLTDELAY.C *****/
```

```
#include <stdio.h>
#include <string.h>
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
{
    int count, dev;
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info below */
    FILE *stream;
    char line[10], *result;

    stream = fopen("posttext.dat", "r");
    result = fgets(line, 10, stream);

    auto_select_display();
    virtual_display(YES);
    get_pair_info("posdelay.dat", &count, &xmin, &xmax, &ymin, &ymax);
    hrange(0.0, xmax);
    vrange(ymin, ymax);
    graph_type(ORDINARY);
    line_connect(NO);
    display_onscreen(YES);
    symbols(SQUARE, DOT, DOT, DOT, DOT, DOT);
    title("Average Delay vs Time", CENTER);

    if (strcmp(line, "Store-and")) {
        haxis_lbl("Simulation Time (slots)", CENTER);
        vaxis_lbl("Average Delay (slots)", CENTER);
    }
    else {
        haxis_lbl("Simulation Time (seconds)", CENTER);
        vaxis_lbl("Average Delay (seconds)", CENTER);
    }
    hlbl_prec(1);
    vlbl_prec(3);
    display_window(0, 900, 0, 643);
    graph_init();
    plot_pairs(1, "posdelay.dat");
    graph_close();
}
```

```
/****** PLTTHRPT.C *****/
```

```
#include <stdio.h>
#include <string.h>
#include "sci-graf.h" /* header file defining sci-graf constants */
```

```
main()
{
    int count, dev;
    double xmin, xmax, ymin, ymax; /* variables used in get_pair_info below */
    FILE *stream;
    char line[10], *result;

    stream = fopen("posttext.dat", "r");
    result = fgets(line, 10, stream);

    auto_select_display();
    virtual_display(YES);
    get_pair_info("posthrpt.dat", &count, &xmin, &xmax, &ymin, &ymax);
    hrange(0.0, xmax);
    vrange(ymin, ymax);
    graph_type(ORDINARY);
    line_connect(NO);
    display_onscreen(YES);
    symbols(SQUARE, DOT, DOT, DOT, DOT, DOT);
    title("Average Throughput vs Time", CENTER);

    if (strcmp(line, "Store-and")) {
        haxis_lbl("Simulation Time (slots)", CENTER);
        vaxis_lbl("Average Throughput", CENTER);
    }
    else {
        haxis_lbl("Simulation Time (seconds)", CENTER);
        vaxis_lbl("Average Throughput", CENTER);
    }
    hlbl_prec(1);
    vlbl_prec(5);
    display_window(0, 900, 0, 643);
    graph_init();
    plot_pairs(1, "posthrpt.dat");
    graph_close();
}
```

```
{***** STARNET.PAS *****}
```

```
program star_network(input, output);
($I sci-graf.inc)
```

```
const
```

```
station_max      = 8;
channel_max      = 16;
session_max      = 64;
queue_max        = 20;
file_size_max    = 100;
default_plot_interval = 10;
dot_display_freq = 1000;
post_text_file   = 'posttext.dat';
plot_disk_file   = 'plotfile.dat';
post_delay_file  = 'posdelay.dat';
post_throughput_file = 'posthrpt.dat';
post_queue_file  = 'posqueue.dat';
input_network_file = 'starnet.tdt';
```

```
type
```

```
str80      = packed array[1..80] of char;
label_name = string[80];
plot_type  = (delay_curve, throughput, queue_size);
node_number = 0..station_max;
session_number = 0..session_max;
channel_number = 0..channel_max;
event_kind  = (arrival, rcv_pkt, departure);
link        = ^event;
event       =
```

```
record
    fptr, bptr : link;
    kind : event_kind;
    time : double;
    arrival_time : double;
    pkt_time : double;
    node_no : node_number;
    session_no : session_number
end;
```

```
var
```

```
origin, session, channel, node, exp_seed      : integer;
pkt_length, pkt_time, max_pkt_time, max_time, service_time : double;
dummy, fixed_pkt_length                       : double;
current_event, base                           : link;
```

```
{ graphics oriented variables }
```

```
network_file, text_file      : text;
plot_file, delay_file, throughput_file, queue_file : text;
plot_yesno, plot_int, plot_session : integer;
plot_channel, plot_interval : integer;
plot_session_in, plot_session_out, plot_channel_in : integer;
plot_channel_out             : integer;
plot_factor                  : double;
plot_switch                   : boolean;
```

```

plot                                     : plot_type;
name1, name2, name3, name4, name5, name6 : str80;

```

```

{ session oriented variables }
session_total      : session_number;
max_traffic        : double;
num_of_arv, num_of_depart : array [session_number] of longint;
mean_pkt_length, traffic_rqmt : array[session_number] of double;
session_rqmt       : array[session_number,1..2] of node_number
average_delay, variance_delay : array [session_number] of double;
last_departure_time : array [session_number] of double;
inter_departure_time : array [session_number] of double;

```

```

{ channel oriented variables }
channel_total      : channel_number;
back_log          : array[channel_number] of integer;
last_arrival_time, wait_time, capacity : array[channel_number] of double;
route             : array[node_number,session_number] of channel_number

```

```

{ node oriented variables }
origin_no, destination_no, node_total : node_number;
node_process_time                     : array[node_number] of double;
link_matrix                          : array[channel_number,1..2] of node_number;
row, column                          : node_number;

```

```

{ variables for random number generation }
glinext, glinextp : integer;
glma              : array [1..55] of double;

```

```

{
the network is characterized by three matrices:

```

1. link_matrix

	from	to	capacity
channel_number	node_no	node_no	bits/sec

2. session_matrix

	origin	destination	traffic_rqmt	mean_pkt_length
session_number	node_no	node_no	packets/sec	bits/packet

3. rotuing_matrix

```

session_number      node_process_time

```

node_number	out_bound_queue	channel_number	seconds
-------------	-----------------	----------------	---------

}

(\$I header.inc)

function ran3(var idum: integer): double;

const

mbig = 4.0e6;
mseed = 1618033.0;
mz = 0.0;
fac = 2.5e-7; (* 1/mbig *)

var

i,ii,k : integer;
mj,mk : double;

begin

if (idum < 0) then

begin

mj := mseed+idum;
if mj>=0.0 then mj := mj-mbig*trunc(mj/mbig)
else mj := mbig-abs(mj)+mbig*trunc(abs(mj)/mbig);

glma[55] := mj;

mk := 1;

for i := 1 to 54 do

begin

ii := 21*i mod 55;
glma[ii] := mk;
mk := mj-mk;
if (mk < mz) then mk := mk+mbig;
mj := glma[ii]

end;

for k := 1 to 4 do

begin

for i := 1 to 55 do
begin
glma[i] := glma[i]-glma[1+((i+30) mod 55)];
if (glma[i] < mz) then glma[i] := glma[i]+mbig
end

end;

glinext := 0;

glinextp := 31;

idum := 1

end;

glinext := glinext+1;

if (glinext = 56) then glinext := 1;

glinextp := glinextp+1;

```

if (glinextp = 56) then glinextp := 1;
mj := glma[glinext]-glma[glinextp];
if (mj < mz) then mj := mj+mbig;
glma[glinext] := mj;
ran3 := mj*fac
end;

procedure str_to_pack(      in_str : label_name;
                           var pack_str : str80      );
var
  max_length, element      : integer;
begin
  max_length := length(in_str);
  for element := 1 to max_length do
    pack_str[element] := in_str[element];
  pack_str[max_length+1] := chr(0);
end;

procedure plot_initialize;
begin
  case plot_int of
    1 : plot := delay_curve;
    2 : plot := throughput;
    3 : plot := queue_size;
  end;

  auto_select_display;
  virtual_display(YES);
  hrange(max_time, 0.0);
  case plot of
    delay_curve :
      begin
        str_to_pack('Delay vs Time', name1);
        str_to_pack('Simulation Time (seconds)', name2);
        str_to_pack('Average Delay (seconds)', name3);
        vrange(queue_max * max_pkt_time, 0.0);
      end;
    throughput :
      begin
        str_to_pack('Throughput vs Time', name1);
        str_to_pack('Simulation Time (seconds)', name2);
        str_to_pack('Average Throughput', name3);
        vrange(1.0, 0.0);
      end;
    queue_size :
      begin
        str_to_pack('Queue Size vs Time', name1);
        str_to_pack('Simulation Time (seconds)', name2);
        str_to_pack('Average Queue Size', name3);
        vrange(queue_max * 1.0, 0.0);
      end;
  end; { end case }
end;

```

```

graph_type(ORDINARY);
line_connect(NO);
display_onscreen(YES);
symbols(DOT, DOT, DOT, DOT, DOT, SQUARE);
hlbl_prec(1);
vlbl_prec(3);
clear_display(NO, YES);

```

```

title(CENTER, name1);
haxis_lbl(CENTER, name2);
vaxis_lbl(CENTER, name3);

```

```

display_window(643, 0, 900, 0);
graph_init;
end;

```

```

procedure plot_graph(disk_file : label_name);
begin
    str_to_pack('', name1);
    str_to_pack('', name2);
    str_to_pack('', name3);
    str_to_pack('', name4);
    str_to_pack('', name5);
    str_to_pack(disk_file, name6);
    plot_pairs(name1, name2, name3, name4, name5, name6, 1);
    clear_display(NO, NO);
    background(NO);
end;

```

```

function expdev(var idum: integer; session: integer): double;
begin
    expdev := -ln( ran3(idum) )/traffic_rqmt[session];
end;

```

```

procedure inter_arrival_time (var next_arrival : double;
                               session : integer);
begin
    next_arrival := expdev(exp_seed, session);
end;

```

```

procedure pkt_distribution(var pkt_size: double; session: integer);
begin
    pkt_size := mean_pkt_length[session];
end;

```

```

procedure generate_event (evkind      : event_kind;
                          current_time: double;
                          arv_time    : double;

```

```

                                pkt_len      : double;
                                node          : node_number;
                                session       : session_number);

var
  event,new_event : link;
  out_bound_queue : channel_number;
  delay : double;

begin
  new(new_event);
  case evkind of
    arrival: { next arrival }
      begin
        inter_arrival_time(delay, session);
        arv_time := current_time + delay;
        new_event^.time := arv_time;
      end;
    departure:
      begin
        new_event^.time := current_time + node_process_time[node];
      end;
    rcv_pkt:
      begin
        out_bound_queue := route[node,session];
        new_event^.time := current_time + node_process_time[node]
          + pkt_len + wait_time[out_bound_queue];
      end;
  end; { end case }

  with new_event^ do
    begin
      kind := evkind;
      arrival_time := arv_time;
      pkt_time := pkt_len;
      node_no := node;
      session_no := session;
    end; { with }
    event := base;
    repeat
      event := event^.bptr
    until new_event^.time >= event^.time;
    new_event^.fptr := event^.fptr;
    new_event^.bptr := event;
    event^.fptr^.bptr := new_event;
    event^.fptr := new_event;
  end; { generate_event }

begin
  { read in network size }
  reset(network_file, input_network_file);
  readln(network_file, node_total);

  { read in plotting switch }

```

```

readln(network_file, plot_session_in);
readln(network_file, plot_session_out);
readln(network_file, plot_channel_in);
readln(network_file, plot_channel_out);
plot_session := 1;
plot_channel := 1;
plot := delay_curve;
plot_switch := TRUE;

{ read in netowrk matrices }
session := 1;
for row := 1 to station_max do { build session matrix from traffic matrix
  for column := 1 to station_max do begin
    readln(network_file, traffic_rqmt[session]);
    if ( (abs(traffic_rqmt[session]) > 1.0e-6 ) and
        (column <= node_total) and (row <= node_total) ) then begin
      session_rqmt[session,1] := row;
      session_rqmt[session,2] := column;
      if ((plot_session_in = row) and (plot_session_out = column))
        then plot_session := session;
      mean_pkt_length[session] := 1.0;
      session := session + 1;
    end;
  end;
session_total := session - 1;

channel_total := 2 * ( node_total-1 );
for channel := 1 to (node_total-1) do begin { build link matrix }
  readln(network_file, capacity[channel]);
  link_matrix[channel,1] := 1;
  link_matrix[channel,2] := channel+1;
  if ((plot_channel_in = link_matrix[channel,1]) and
      (plot_channel_out = link_matrix[channel,2]))
    then plot_channel := channel;
end;

{ read in dummy entries }
for channel := node_total to (channel_max div 2) do
  readln(network_file, dummy);

{ continue to build link matrix }
for channel := node_total to channel_total do begin
  readln(network_file, capacity[channel]);
  link_matrix[channel,2] := 1;
  link_matrix[channel,1] := channel - node_total+2;
  if ((plot_channel_in = link_matrix[channel,1]) and
      (plot_channel_out = link_matrix[channel,2]))
    then plot_channel := channel;
end;

{ read in dummy entries }
for channel := node_total to (channel_max div 2) do
  readln(network_file, dummy);

for node := 2 to node_total do { build routing matrix for star only }

```

```

begin
    node_process_time[node] := 0.0;
    for session := 1 to session_total do
        route[node,session] := node + node_total -2;
    end;
node_process_time[1] := 0.0;
for session := 1 to session_total do
    route[1,session] := session_rqmt[session,2]-1;

readln(network_file, max_time);
readln(network_file, plot_int);
readln(network_file, plot_yesno);

close(network_file);

( echo the three matrices )
rewrite(text_file, post_text_file);
writeln(text_file, 'Store-and-Forward Network - Star Topology');
writeln(text_file);
writeln(text_file, 'Input');
writeln(text_file, '====');
writeln(text_file, ' Routing Matrix :');
for node := 1 to node_total do { routing matrix }
    begin
        for session := 1 to session_total do
            write(text_file, route[node,session]:3);
            writeln(text_file, node_process_time[node]:10:5);
        end;
    writeln(text_file);
    writeln(text_file, ' Session Matrix :');
    for session := 1 to session_total do { session matrix }
        writeln(text_file, session_rqmt[session,1]:3, session_rqmt[session,2]:3,
            traffic_rqmt[session]:10:5, mean_pkt_length[session]:10:5);
    writeln(text_file);
    writeln(text_file, ' Link Matrix :');
    for channel := 1 to channel_total do { link matrix }
        writeln(text_file, link_matrix[channel,1]:3, link_matrix[channel,2]:3,
            capacity[channel]:10:5);
    writeln(text_file);
    writeln(text_file, 'Simulation Results of Session',plot_session:4);
    writeln(text_file, 'Simulated Time      Average Delay      Throughput');
    writeln(text_file, '=====');

( initialize variables )
exp_seed := -1;
for channel := 1 to channel_total do
    begin
        wait_time[channel] := 0.0;
        last_arrival_time[channel] := 0.0;
        back_log[channel] := 0;
    end;
max_traffic := 0.0;
for session := 1 to session_total do
    begin
        last_departure_time[session] := 0.0;

```

```

    average_delay[session] := 0.0;
    variance_delay[session] := 0.0;
    num_of_arv[session] := 0;
    num_of_depart[session] := 0;
    if max_traffic <= traffic_rqmt[session] then
        max_traffic := traffic_rqmt[session];
    end;
    plot_factor := (max_traffic * max_time)
                  / (file_size_max * default_plot_interval);
    plot_interval := default_plot_interval;
    if plot_factor > 1.0 then
        plot_interval := default_plot_interval * trunc(plot_factor + 1.0);

    rewrite(delay_file, post_delay_file);
    rewrite(throughput_file, post_throughput_file);
    rewrite(queue_file, post_queue_file);
    rewrite(plot_file, plot_disk_file);

    case plot_yesno of
        1 : plot_switch := TRUE;
        0 : plot_switch := FALSE;
    end; { end case }
    IF plot_int = 0 THEN plot_switch := FALSE;
    if (not plot_switch) then writeln('Running...');

    { create an empty event ring }
    new(base);
    with base^ do
        begin
            fptr := base;
            bptr := base;
            time := 0.0
        end;

    { generate arrival for each session }
    max_pkt_time := 0.0;
    for session := 1 to session_total do
        begin
            origin := session_rqmt[session,1];
            channel := route[origin,session];
            pkt_distribution(pkt_length, session);
            pkt_time := pkt_length / capacity[channel];
            if max_pkt_time <= pkt_time then max_pkt_time := pkt_time;
            generate_event(arrival, 0.0, 0.0, pkt_time, origin, session);
        end;

    if plot_switch then plot_initialize;

    repeat
        current_event := base^.fptr;
        with current_event^ do
            case kind of
                arrival:
                    begin
                        num_of_arv[session_no] := num_of_arv[session_no] + 1;

```

```

generate_event(arrival, time, arrival_time, pkt_time, node_no,
session_no);
channel := route[node_no,session_no];
if back_log[channel] = 0 then
begin
    wait_time[channel] := 0.0;
    last_arrival_time[channel] := arrival_time;
    back_log[channel] := back_log[channel] + 1;
end
else
begin
    wait_time[channel] := wait_time[channel] + pkt_time
        - ( arrival_time - last_arrival_time[channel] );
    last_arrival_time[channel] := arrival_time;
    if wait_time[channel] <= 0.0 then
        wait_time[channel] := 0.0;
    back_log[channel] := back_log[channel] + 1;
end;
generate_event(rcv_pkt, time, arrival_time, pkt_time, node_no,
session_no);
if ((num_of_arv[plot_session] mod dot_display_freq = 0)
and (not plot_switch)) then write('.');
end;
departure:
begin
    num_of_depart[session_no] := num_of_depart[session_no] + 1;
    service_time := current_event^.time - current_event^.arrival_time
    average_delay[session_no] := average_delay[session_no]+service_
time;
    variance_delay[session_no] := variance_delay[session_no]
        + service_time * service_time;
    inter_departure_time[session_no] := current_event^.time
        - last_departure_time[session_no];
    last_departure_time[session_no] := current_event^.time;

    if (num_of_depart[session_no] mod plot_interval) = 0 then
begin
    if plot_session = session_no then
begin
        writeln(delay_file, current_event^.time:10:3,
            average_delay[session_no]/num_of_depart[session_no]:10:5)
        writeln(throughput_file, current_event^.time:10:3,
            num_of_depart[session_no]/num_of_arv[session_no]:10:5);
        writeln(text_file, current_event^.time:10:3,
            average_delay[session_no]/num_of_depart[session_no]:20:5,
            num_of_depart[session_no]/num_of_arv[session_no]:18:5);
    end;
    if plot_switch then
begin
        reset(plot_file, plot_disk_file);
        case plot of
            delay_curve: if plot_session = session_no then
                writeln(plot_file, current_event^.time:10:3,

```

```

        average_delay[session_no]/num_of_depart[session_no]:10:5);
        throughput: if plot_session = session_no then
            writeln(plot_file, current_event^.time:10:3,
                num_of_depart[session_no]/num_of_arv[session_no]:10:5);
        end; { end case }
        close(plot_file);
        plot_graph(plot_disk_file);
    end;
end;
end;

rcv_pkt:
begin
    channel := route[node_no,session_no];
    back_log[channel] := back_log[channel] - 1;

    if(num_of_depart[session_no] mod plot_interval) = 0 then
        begin
            if plot_channel = channel then
                writeln(queue_file, current_event^.time:10:5,
                    back_log[channel]:5);

                if plot_switch then
                    begin
                        reset(plot_file, plot_disk_file);
                        if plot = queue_size then
                            if plot_channel = channel then
                                writeln(plot_file, current_event^.time:10:5,
                                    back_log[channel]:5);
                                close(plot_file);
                                plot_graph(plot_disk_file);
                            end;
                        end;
                    end;

                node := link_matrix[channel,2];
                if node = session_rqmt[session_no,2]
                then
                    { final destination }
                    begin
                        generate_event(departure, time, arrival_time, pkt_time, node,
session_no)
                    end
                else
                    { route to next node }
                    begin
                        channel := route[node,session_no];
                        if back_log[channel] = 0 then
                            begin
                                wait_time[channel] := 0.0;
                                back_log[channel] := back_log[channel] + 1;
                                last_arrival_time[channel] := time;
                            end
                        else

```

```

begin
    wait_time[channel] := wait_time[channel] + pkt_time
                        - ( time - last_arrival_time[channel] );
    last_arrival_time[channel] := time;
    if wait_time[channel] <= 0.0 then
        wait_time[channel] := 0.0;
        back_log[channel] := back_log[channel] + 1;
    end;
    generate_event(rcv_pkt, time, arrival_time, pkt_time, node,
session_no);
end;
end;
end;
base^.fptr := current_event^.fptr;
current_event^.fptr^.bptr := base;
dispose(current_event);
until ((base^.fptr^.time >= max_time)
        or (back_log[plot_session] >= queue_max));

if plot_switch then graph_close;

close(delay_file);
close(throughput_file);
close(queue_file);

{ collect statistics }
writeln(text_file);
writeln(text_file, 'Final Statistics:');
writeln(text_file, '=====');
writeln(text_file, ' maximum simulation time: ', max_time:10:2, ' seconds')
for session := 1 to session_total do
    writeln(text_file, ' average arrival rate of session', session:5, ' is',
        traffic_rqmt[session]:10:5, ' packets per second');
    writeln(text_file, '
                                number of
                                delay
(seconds)');
    writeln(text_file, ' session arrival departure average
variance');
    for session := 1 to session_total do
        begin
            if num_of_depart[session] = 0
            then average_delay[session] := 0.0
            else average_delay[session] := average_delay[session]
                / num_of_depart[session];

            if num_of_depart[session] <= 1
            then variance_delay[session] := 0.0
            else variance_delay[session] := (variance_delay[session]
- num_of_depart[session] * average_delay[session] * average_delay[ses-
sion])
                / (num_of_depart[session] - 1);

            writeln(text_file, session:10,
                num_of_arv[session]:10, num_of_depart[session]:10,
                average_delay[session]:15:5, variance_delay[session]:15:5);
        end;
    end;
close(text_file);
end.

```



```
{***** TREE.PAS *****}
```

```
PROGRAM tree_algorithm (input, output);
($I sci-graf.inc)
```

```
CONST
```

```
channel_max      = 10;
station_max      = 20;
updown_max       = 10;
file_size_max    = 100;
default_plot_interval = 10;
dot_display_freq = 1000;
post_text_file   = 'posttext.dat';
plot_disk_file   = 'plotfile.dat';
post_delay_file  = 'posdelay.dat';
post_throughput_file = 'posthrpt.dat';
post_queue_file  = 'posqueue.dat';
input_file       = 'tree.tdt';
```

```
TYPE
```

```
str80            = packed array[1..80] of char;
label_name       = string[80];
plot_type        = (delay_curve, throughput, queue_size);
channel_number   = 0..channel_max;
station_number   = 0..station_max;
updown_number    = 0..updown_max;
event_kind       = (arrival, transmission, wait_for_clear, resolution,
                    departure, END_of_slot, ck_glb_count, tx_after_blk);
coin             = (head, tail);
channel_state    = (idle, busy, collision);
link             = ^event;
event            =
```

```
RECORD
```

```
    fptr, bptr : link;
    kind : event_kind;
    time : double;
    arrival_time : double;
    updown_no : updown_number;
    channel_no : channel_number;
    station_no : station_number
```

```
END;
```

```
VAR
```

```
num_of_arv, num_of_depart : ARRAY [station_number] OF longint;
mean_time                 : ARRAY [event_kind] OF double;
global_count              : ARRAY [channel_number] OF integer;
state                    : ARRAY [channel_number] OF chan-
nel_state;
backlog, user, population, station, channel, coin_seed, exp_seed : integer;
input_rate, max_traffic, max_time, service_time : double;
average_delay, variance_delay : ARRAY [station_number] OF double;
last_departure_time, inter_departure_time : ARRAY [station_number] OF
double;
current_event, base      : link;
event_index              : event_kind;
```

```

station_total      : station_number;
channel_total      : channel_number;
coin_face          : coin;
glinext, glinextp  : integer;
glma               : ARRAY [1..55] OF double;

```

```

( graphics oriented variables )
configuration_file, text_file      : text;
plot_file, delay_file, throughput_file, queue_file : text;
plot_yesno, plot_int               : integer;
plot_channel, plot_interval        : integer;
plot_factor                        : double;
plot_switch                        : boolean;
plot                               : plot_type;
name1, name2, name3, name4, name5, name6 : str80;

```

```

($I header.inc)

```

```

FUNCTION ran3(VAR idum: integer): double;

```

```

const

```

```

    mbig      = 4.0e6;
    mseed     = 1618033.0;
    mz        = 0.0;
    fac       = 2.5e-7; (* 1/mbig *)

```

```

var

```

```

    i,ii,k    : integer;
    mj,mk     : double;

```

```

BEGIN

```

```

    if (idum < 0) then

```

```

        BEGIN

```

```

            mj := mseed+idum;
            if mj>=0.0 then mj := mj-mbig*trunc(mj/mbig)
                else mj := mbig-abs(mj)+mbig*trunc(abs(mj)/mbig);

```

```

            glma[55] := mj;

```

```

            mk := 1;

```

```

            for i := 1 to 54 do

```

```

                BEGIN

```

```

                    ii := 21*i mod 55;

```

```

                    glma[ii] := mk;

```

```

                    mk := mj-mk;

```

```

                    if (mk < mz) then mk := mk+mbig;

```

```

                    mj := glma[ii]

```

```

                END;

```

```

            for k := 1 to 4 do

```

```

                BEGIN

```

```

                    for i := 1 to 55 do

```

```

                        BEGIN

```

```

                            glma[i] := glma[i]-glma[1+((i+30) mod 55)];

```

```

                            if (glma[i] < mz) then glma[i] := glma[i]+mbig

```

```

                        END

```

```

                    END;

```

```

            glinext := 0;

```

```

            glinextp := 31;

```

```

            idum := 1

```

```

    END;
    glinext := glinext+1;
    if (glinext = 56) then glinext := 1;
    glinextp := glinextp+1;
    if (glinextp = 56) then glinextp := 1;
    mj := glma[glinext]-glma[glinextp];
    if (mj < mz) then mj := mj+mbig;
    glma[glinext] := mj;
    ran3 := mj*fac
END;

procedure str_to_pack(      in_str : label_name;
                           var pack_str : str80      );
var
    max_length, element    : integer;

BEGIN
    max_length := length(in_str);
    for element := 1 to max_length do
        pack_str[element] := in_str[element];
    pack_str[max_length+1] := chr(0);
END;

procedure plot_initialize;
BEGIN
    case plot_int of
        1 : plot := delay_curve;
        2 : plot := throughput;
        3 : plot := queue_size;
    END;

    auto_select_display;
    virtual_display(YES);
    hrange(max_time, 0.0);
    case plot of
        delay_curve :
            BEGIN
                str_to_pack('Delay vs Time', name1);
                str_to_pack('Simulation Time (slots)',name2);
                str_to_pack('Average Delay (slots)',name3);
                vrange(updown_max*1.0, 0.0);
            END;
        throughput :
            BEGIN
                str_to_pack('Throughput vs Time', name1);
                str_to_pack('Simulation Time (slots)',name2);
                str_to_pack('Average Throughput',name3);
                vrange(1.0, 0.0);
            END;
        queue_size :
            BEGIN
                str_to_pack('Queue Size vs Time', name1);
                str_to_pack('Simulation Time (slots)',name2);

```

```

        str_to_pack('Average Queue Size',name3);
        vrange(10.0, 0.0);
    END;
    END; { END case }
graph_type(ORDINARY);
line_connect(NO);
display_onscreen(YES);
symbols(DOT, DOT, DOT, DOT, DOT, SQUARE);
hlbl_prec(1);
vlbl_prec(3);
clear_display(NO, YES);

title(CENTER, name1);
haxis_lbl(CENTER, name2);
vaxis_lbl(CENTER, name3);

display_window(643, 0, 900, 0);
graph_init;
END;

procedure plot_graph(disk_file : label_name);
BEGIN
    str_to_pack('',name1);
    str_to_pack('',name2);
    str_to_pack('',name3);
    str_to_pack('',name4);
    str_to_pack('',name5);
    str_to_pack(disk_file,name6);
    plot_pairs(name1, name2, name3, name4, name5, name6, 1);
    clear_display(NO, NO);
    background(NO);
END;

FUNCTION expdev(VAR idum: integer): double;
BEGIN
    expdev := -ln(ran3(idum))/mean_time[arrival]
END;

PROCEDURE inter_arrival_time (VAR next_arrival: double);
BEGIN
    next_arrival := expdev(exp_seed);
END;

PROCEDURE generate_event (evkind      : event_kind;
                          current_time: double;
                          arv_time    : double;
                          updown      : updown_number;
                          channel      : channel_number;

```

```

                                station      : station_number);

VAR
  event,new_event : link;
  delay : double;
BEGIN
  new(new_event);
  CASE evkind OF
    arrival: { next arrival }
      BEGIN
        inter_arrival_time(delay);
        arv_time := current_time + delay;
        new_event^.time := arv_time;
      END;
    transmission: { time to next slot }
      BEGIN
        new_event^.time := trunc(current_time) + 1.0001;
      END;
    resolution: { resolution is scheduled at the middle of the slot }
      BEGIN
        new_event^.time := current_time + 0.5;
      END;
    wait_for_clear: { transmissions that flip tails }
      BEGIN
        new_event^.time := trunc(current_time + 1.5) - 0.0001;
      END;
    departure: { total packet transmission time is one slot }
      BEGIN
        new_event^.time := current_time + 0.4999;
      END;
    END_of_slot:
      BEGIN
        new_event^.time := current_time + 1.0;
      END;
    ck_glb_count:
      BEGIN
        new_event^.time := trunc(current_time) + 1.0001;
      END;
    tx_after_blk:
      BEGIN
        new_event^.time := current_time;
      END;
  END; { END CASE }

  WITH new_event^ DO
    BEGIN
      kind := evkind;
      arrival_time := arv_time;
      updown_no := updown;
      channel_no := channel;
      station_no := station;
    END; { with }
    event := base;
    REPEAT
      event := event^.bptr
    UNTIL new_event^.time >= event^.time;

```

```

new_event^.fptr := event^.fptr;
new_event^.bptr := event;
event^.fptr^.bptr := new_event;
event^.fptr := new_event;
END; { generate_event }

```

```

BEGIN

```

```

{ initialize variables }

```

```

exp_seed := -1;
coin_seed := -1;
plot := delay_curve;
plot_switch := TRUE;

```

```

reset(configuration_file, input_file);
readln(configuration_file, population);
readln(configuration_file, channel_total);

```

```

station_total := 1;
channel_total := 1;
FOR user := 1 TO station_max DO

```

```

    BEGIN

```

```

        readln(configuration_file, input_rate);
        mean_time[arrival] := mean_time[arrival] + input_rate;

```

```

    END;

```

```

readln(configuration_file, max_time);
readln(configuration_file, plot_int);
readln(configuration_file, plot_yesno);
close(configuration_file);

```

```

max_traffic := mean_time[arrival];
plot_factor := (max_traffic * max_time)
                / (file_size_max * default_plot_interval);
plot_interval := default_plot_interval;
if plot_factor > 1.0 then
    plot_interval := default_plot_interval * trunc(plot_factor + 1.0);

```

```

rewrite(text_file, post_text_file);
rewrite(delay_file, post_delay_file);
rewrite(throughput_file, post_throughput_file);
rewrite(queue_file, post_queue_file);
rewrite(plot_file, plot_disk_file);
writeln(text_file, 'Multiple Access Protocol - Tree (single channel)');
writeln(text_file);
writeln(text_file, 'Simulation Results:');
writeln(text_file, 'Simulated Time      Average Delay      Throughput');
writeln(text_file, '=====');

```

```

case plot_yesno of

```

```

    1 : plot_switch := TRUE;
    0 : plot_switch := FALSE;

```

```

END; { END case }

```

```

IF plot_int = 0 THEN plot_switch := FALSE;
if (not plot_switch) then writeln('Running...');

```

```

{ create an empty event ring }
new(base);
WITH base^ DO
  BEGIN
    fptr := base;
    bptr := base;
    time := 0.0
  END;

{ generate arrival at each station }
FOR station := 1 TO station_total DO
  BEGIN
    last_departure_time[station] := 0.0;
    average_delay[station] := 0.0;
    variance_delay[station] := 0.0;
    num_of_arv[station] := 0;
    num_of_depart[station] := 0;
    FOR channel := 1 TO channel_total DO
      generate_event(arrival, 0.0, 0.0, 0, channel, station);
    END;

{ generate initial conditions for each channel }
FOR channel := 1 TO channel_total DO
  BEGIN
    generate_event(END_of_slot, 0.0, 0.0, 0, channel, 0);
    state[channel] := idle;
    global_count[channel] := 0;
  END;

if plot_switch then plot_initialize;

REPEAT
  current_event := base^.fptr;
  WITH current_event^ DO
    CASE kind OF
      arrival:
        BEGIN
          num_of_arv[station_no] := num_of_arv[station_no] + 1;
          generate_event(arrival, time, arrival_time, updown_no, channel_no
station_no);
          generate_event(ck_glb_count, time, arrival_time, updown_no,
channel_no, station_no);
        END;
      ck_glb_count:
        BEGIN
          IF global_count[channel_no] = 0
            THEN generate_event(tx_after_blk, time, arrival_time, updown_no
channel_no, station_no)
            ELSE generate_event(ck_glb_count, time, arrival_time, updown_no
channel_no, station_no);
          END;
          tx_after_blk, transmission:
            BEGIN
              CASE state[channel_no] OF

```

```

        idle: state[channel_no] := busy;
        busy: state[channel_no] := collision;
        collision: state[channel_no] := collision;
    END; { case }
    generate_event(resolution, time, arrival_time, updown_no,
channel_no, station_no);
    END;
    wait_for_clear:
    BEGIN
        IF state[channel_no] = collision
            THEN updown_no := updown_no + 1
            ELSE updown_no := updown_no - 1;
        IF updown_no = 0
            THEN generate_event(transmission, time, arrival_time, updown_no
channel_no, station_no)
            ELSE generate_event(wait_for_clear, time, arrival_time, up-
down_no, channel_no, station_no);
        END;
    resolution:
    BEGIN
        IF state[channel_no] = collision
            THEN
                BEGIN
                    IF global_count[channel_no] = 0
                        THEN global_count[channel_no] := global_count[channel_no]
+ 1;

                    IF updown_no = 0 THEN updown_no := updown_no + 1;
                    IF ran3(coin_seed) >= 0.5 THEN coin_face := tail
                        ELSE coin_face := head;
                    IF coin_face = tail
                        THEN
                            BEGIN { transmissions that flip tails }
                                generate_event(wait_for_clear, time, arrival_time,
updown_no, channel_no, station_no)
                            END
                        ELSE
                            BEGIN { transmissions that flip heads go on next slot }
                                generate_event(transmission, time, arrival_time,
updown_no, channel_no, station_no)
                            END;
                        END
                    ELSE
                        BEGIN
                            generate_event(departure, time, arrival_time, updown_no,
channel_no, station_no);
                        END;
                    END;
    END;
    departure:
    BEGIN
        num_of_depart[station_no] := num_of_depart[station_no] + 1;
        service_time := current_event^.time - current_event^.arrival_time
        average_delay[station_no] := average_delay[station_no] + service_
time;
        variance_delay[station_no] := variance_delay[station_no]
            + service_time * service_time;

```

```

if (num_of_depart[station_no] mod plot_interval) = 0 then
BEGIN
  writeln(delay_file, current_event^.time:10:3,
    average_delay[station_no]/num_of_depart[station_no]:10:5)
  writeln(throughput_file, current_event^.time:10:3,
    num_of_depart[station_no]/current_event^.time:10:5);
  writeln(text_file, current_event^.time:10:3,
    average_delay[station_no]/num_of_depart[station_no]:20:5,
    num_of_depart[station_no]/current_event^.time:18:5);
  if plot_switch then
    BEGIN
      reset(plot_file, plot_disk_file);
      case plot of
        delay_curve:
          writeln(plot_file, current_event^.time:10:3,
            average_delay[station_no]/num-
_of_depart[station_no]:10:5);
          throughput:
            writeln(plot_file, current_event^.time:10:3,
              num_of_depart[station_no]/current_event^.time:10:5);
        END; { END case }
      close(plot_file);
      plot_graph(plot_disk_file);
    END;
  END;
  inter_departure_time[station_no] := current_event^.time
    - last_departure_time[station_no]
  last_departure_time[station_no] := current_event^.time;
END;
END_of_slot:
BEGIN
  IF state[channel_no] = collision
    THEN global_count[channel_no] := global_count[channel_no] + 1
    ELSE IF global_count[channel_no] > 0
      THEN global_count[channel_no] := global_count[channel_no]
1;
    state[channel_no] := idle;
    generate_event(END_of_slot, time, arrival_time, updown_no,
channel_no, station_no);
    if (((trunc(current_event^.time) mod dot_display_freq) = 0)
      and (not plot_switch)) then write('.');
    END;
  END;
  base^.fptr := current_event^.fptr;
  current_event^.fptr^.bptr := base;
  backlog := current_event^.updown_no;
  dispose(current_event);
  UNTIL ((base^.fptr^.time >= max_time) or (backlog >= updown_max));

  if plot_switch then graph_close;

  close(delay_file);
  close(throughput_file);
  close(queue_file);

```

```

{ collect statistics }
writeln(text_file);
writeln(text_file, 'Final Statistics:');
writeln(text_file, '=====');
writeln(text_file, ' Maximum simulation slots : ', max_time:10:2);
writeln(text_file, ' Number of Channels : ', channel_total:5);
writeln(text_file, ' Average arrival rate : ',
      mean_time[arrival]:10:5, ' packets per slot');
writeln(text_file, '
      Number of
      Delay
(slots)');
writeln(text_file, ' Station arrival departure average
variance');
FOR station := 1 TO station_total DO
  BEGIN
    IF num_of_depart[station] = 0
      THEN average_delay[station] := 0.0
      ELSE average_delay[station] := average_delay[station]
        / num_of_depart[station];

    IF num_of_depart[station] <= 1
      THEN variance_delay[station] := 0.0
      ELSE variance_delay[station] := (variance_delay[station]
- num_of_depart[station] * average_delay[station] * average_delay[sta-
tion])
        / (num_of_depart[station] - 1);

    writeln(text_file, station:10,
      num_of_arv[station]:10, num_of_depart[station]:10,
      average_delay[station]:15:5, variance_delay[station]:15:5);

  END;
close(text_file);
END.

```

```
{***** UNILoop.PAS *****}
```

```
program uni_directional_loop_network(input, output);
{$I sci-graf.inc}
```

```
const
```

```
station_max      = 8;
channel_max      = 8;
session_max      = 64;
queue_max        = 20;
file_size_max    = 100;
default_plot_interval = 10;
dot_display_freq = 1000;
post_text_file   = 'posttext.dat';
plot_disk_file   = 'plotfile.dat';
post_delay_file  = 'posdelay.dat';
post_throughput_file = 'posthrpt.dat';
post_queue_file  = 'posqueue.dat';
input_network_file = 'uniloop.tdt';
```

```
type
```

```
str80      = packed array[1..80] of char;
label_name = string[80];
plot_type  = (delay_curve, throughput, queue_size);
node_number = 0..station_max;
session_number = 0..session_max;
channel_number = 0..channel_max;
event_kind  = (arrival, rcv_pkt, departure);
link        = ^event;
event       =
    record
        fptr, bptr : link;
        kind : event_kind;
        time : double;
        arrival_time : double;
        pkt_time : double;
        node_no : node_number;
        session_no : session_number
    end;
```

```
var
```

```
origin, session, channel, node, exp_seed      : integer;
pkt_length, pkt_time, max_pkt_time, max_time, service_time : double;
dummy, fixed_pkt_length                       : double;
current_event, base                           : link;
```

```
{ graphics oriented variables }
```

```
network_file, text_file      : text;
plot_file, delay_file, throughput_file, queue_file      : text;
plot_yesno, plot_int, plot_session      : integer;
plot_channel, plot_interval      : integer;
plot_session_in, plot_session_out, plot_channel_in      : integer;
plot_channel_out      : integer;
plot_factor      : double;
plot_switch      : boolean;
```

一、
 二、
 三、
 四、
 五、
 六、
 七、
 八、
 九、
 十、
 十一、
 十二、
 十三、
 十四、
 十五、
 十六、
 十七、
 十八、
 十九、
 二十、

1

1

1

1

1

10

■

1

1

1

1

1

node_number

out_bound_queue channel_number

seconds

}

(\$I header.inc)

function ran3(var idum: integer): double;

const

mbig = 4.0e6;
mseed = 1618033.0;
mz = 0.0;
fac = 2.5e-7; (* 1/mbig *)

var

i,ii,k : integer;
mj,mk : double;

begin

if (idum < 0) then

begin

mj := mseed+idum;
if mj>=0.0 then mj := mj-mbig*trunc(mj/mbig)
else mj := mbig-abs(mj)+mbig*trunc(abs(mj)/mbig);

glma[55] := mj;

mk := 1;

for i := 1 to 54 do

begin

ii := 21*i mod 55;
glma[ii] := mk;
mk := mj-mk;
if (mk < mz) then mk := mk+mbig;
mj := glma[ii]

end;

for k := 1 to 4 do

begin

for i := 1 to 55 do

begin

glma[i] := glma[i]-glma[1+((i+30) mod 55)];
if (glma[i] < mz) then glma[i] := glma[i]+mbig

end

end;

glinext := 0;

glinextp := 31;

idum := 1

end;

glinext := glinext+1;

if (glinext = 56) then glinext := 1;

glinextp := glinextp+1;

```

if (glinextp = 56) then glinextp := 1;
mj := glma[glinext]-glma[glinextp];
if (mj < mz) then mj := mj+mbig;
glma[glinext] := mj;
ran3 := mj*fac
end;

```

```

procedure str_to_pack(      in_str : label_name;
                           var pack_str : str80      );
var
  max_length, element      : integer;

```

```

begin
  max_length := length(in_str);
  for element := 1 to max_length do
    pack_str[element] := in_str[element];
  pack_str[max_length+1] := chr(0);
end;

```

```

procedure plot_initialize;
begin
  case plot_int of
    1 : plot := delay_curve;
    2 : plot := throughput;
    3 : plot := queue_size;
  end;

```

```

auto_select_display;
virtual_display(YES);
hrange(max_time, 0.0);
case plot of
  delay_curve :
    begin
      str_to_pack('Delay vs Time', name1);
      str_to_pack('Simulation Time (seconds)', name2);
      str_to_pack('Average Delay (seconds)', name3);
      vrange(queue_max * max_pkt_time, 0.0);
    end;
  throughput :
    begin
      str_to_pack('Throughput vs Time', name1);
      str_to_pack('Simulation Time (seconds)', name2);
      str_to_pack('Average Throughput', name3);
      vrange(1.0, 0.0);
    end;
  queue_size :
    begin
      str_to_pack('Queue Size vs Time', name1);
      str_to_pack('Simulation Time (seconds)', name2);
      str_to_pack('Average Queue Size', name3);
      vrange(queue_max*1.0, 0.0);
    end;
end; { end case }

```

```

graph_type(ORDINARY);
line_connect(NO);
display_onscreen(YES);
symbols(DOT, DOT, DOT, DOT, DOT, SQUARE);
hlbl_prec(1);
vlbl_prec(3);
clear_display(NO, YES);

title(CENTER, name1);
haxis_lbl(CENTER, name2);
vaxis_lbl(CENTER, name3);

display_window(643, 0, 900, 0);
graph_init;
end;

procedure plot_graph(disk_file : label_name);
begin
    str_to_pack('', name1);
    str_to_pack('', name2);
    str_to_pack('', name3);
    str_to_pack('', name4);
    str_to_pack('', name5);
    str_to_pack(disk_file, name6);
    plot_pairs(name1, name2, name3, name4, name5, name6, 1);
    clear_display(NO, NO);
    background(NO);
end;

function expdev(var idum: integer; session: integer): double;
begin
    expdev := -ln( ran3(idum) )/traffic_rqmt[session];
end;

procedure inter_arrival_time (var next_arrival : double;
                               session : integer);
begin
    next_arrival := expdev(exp_seed, session);
end;

procedure pkt_distribution(var pkt_size: double; session: integer);
begin
    pkt_size := mean_pkt_length[session];
end;

procedure generate_event (evkind      : event_kind;
                          current_time: double;
                          arv_time    : double;

```

```

                                pkt_len      : double;
                                node          : node_number;
                                session       : session_number);
var
  event,new_event : link;
  out_bound_queue : channel_number;
  delay : double;

begin
  new(new_event);
  case evkind of
    arrival: { next arrival }
      begin
        inter_arrival_time(delay, session);
        arv_time := current_time + delay;
        new_event^.time := arv_time;
      end;
    departure:
      begin
        new_event^.time := current_time + node_process_time[node];
      end;
    rcv_pkt:
      begin
        out_bound_queue := route[node,session];
        new_event^.time := current_time + node_process_time[node]
          + pkt_len + wait_time[out_bound_queue];
      end;
  end; { end case }

  with new_event^ do
    begin
      kind := evkind;
      arrival_time := arv_time;
      pkt_time := pkt_len;
      node_no := node;
      session_no := session;
    end; { with }
    event := base;
    repeat
      event := event^.bptr
    until new_event^.time >= event^.time;
    new_event^.fptr := event^.fptr;
    new_event^.bptr := event;
    event^.fptr^.bptr := new_event;
    event^.fptr := new_event;
  end; { generate_event }

begin
  { read in network size }
  reset(network_file, input_network_file);
  readln(network_file, node_total);

  { read in plotting switch }
  readln(network_file, plot_session_in);

```

```

readln(network_file, plot_session_out);
readln(network_file, plot_channel_in);
readln(network_file, plot_channel_out);
plot_session := 1;
plot_channel := 1;
plot := delay_curve;
plot_switch := TRUE;

{ read in network matrices }
session := 1;
for row := 1 to station_max do { build session matrix from traffic matrix
  for column := 1 to station_max do begin
    readln(network_file, traffic_rqmt[session]);
    if (abs(traffic_rqmt[session]) > 1.0e-6 ) then begin
      session_rqmt[session,1] := row;
      session_rqmt[session,2] := column;
      if ((plot_session_in = row) and (plot_session_out = column))
        then plot_session := session;
      mean_pkt_length[session] := 1.0;
      session := session + 1;
    end;
  end;
session_total := session - 1;

channel_total := node_total;
for channel := 1 to channel_total do begin { build link matrix }
  readln(network_file, capacity[channel]);
  link_matrix[channel,1] := channel;
  link_matrix[channel,2] := (channel mod node_total) + 1;
  if ((plot_channel_in = link_matrix[channel,1]) and
    (plot_channel_out = link_matrix[channel,2]))
    then plot_channel := channel;
end;

{ read in dummy entries }
if (channel_total < channel_max) then
  for channel := (channel_total + 1) to channel_max do
    readln(network_file, dummy);

for node := 1 to node_total do { build routing matrix for uniloop only }
  begin
    node_process_time[node] := 0.0;
    for session := 1 to session_total do
      route[node,session] := link_matrix[node,1];
    end;

readln(network_file, max_time);
readln(network_file, plot_int);
readln(network_file, plot_yesno);

close(network_file);

{ echo the three matrices }
rewrite(text_file, post_text_file);
writeln(text_file, 'Store-and-Forward Network - Uni-Directional Loop');

```

```

writeln(text_file);
writeln(text_file, 'Input');
writeln(text_file, '====');
writeln(text_file, ' Routing Matrix :');
for node := 1 to node_total do { routing matrix }
  begin
    for session := 1 to session_total do
      write(text_file, route[node,session]:3);
      writeln(text_file, node_process_time[node]:10:5);
    end;
  writeln(text_file);
  writeln(text_file, ' Session Matrix :');
  for session := 1 to session_total do { session matrix }
    writeln(text_file, session_rqmt[session,1]:3, session_rqmt[session,2]:3,
      traffic_rqmt[session]:10:5, mean_pkt_length[session]:10:5);
  writeln(text_file);
  writeln(text_file, ' Link Matrix :');
  for channel := 1 to channel_total do { link matrix }
    writeln(text_file, link_matrix[channel,1]:3, link_matrix[channel,2]:3,
      capacity[channel]:10:5);
  writeln(text_file);
  writeln(text_file, 'Simulation Results of Session',plot_session:4);
  writeln(text_file, 'Simulated Time      Average Delay      Throughput');
  writeln(text_file, '=====');

{ initialize variables }
exp_seed := -1;
for channel := 1 to channel_total do
  begin
    wait_time[channel] := 0.0;
    last_arrival_time[channel] := 0.0;
    back_log[channel] := 0;
  end;
max_traffic := 0.0;
for session := 1 to session_total do
  begin
    last_departure_time[session] := 0.0;
    average_delay[session] := 0.0;
    variance_delay[session] := 0.0;
    num_of_arv[session] := 0;
    num_of_depart[session] := 0;
    if max_traffic <= traffic_rqmt[session] then
      max_traffic := traffic_rqmt[session];
    end;
  plot_factor := (max_traffic * max_time)
    / (file_size_max * default_plot_interval);
  plot_interval := default_plot_interval;
  if plot_factor > 1.0 then
    plot_interval := default_plot_interval * trunc(plot_factor + 1.0);

  rewrite(delay_file, post_delay_file);
  rewrite(throughput_file, post_throughput_file);
  rewrite(queue_file, post_queue_file);
  rewrite(plot_file, plot_disk_file);

```

```

case plot_yesno of
  1 : plot_switch := TRUE;
  0 : plot_switch := FALSE;
end; { end case }
if plot_int = 0 THEN plot_switch := FALSE;
if (not plot_switch) then writeln('Running...');

{ create an empty event ring }
new(base);
with base^ do
  begin
    fptr := base;
    bptr := base;
    time := 0.0
  end;

{ generate arrival for each session }
max_pkt_time := 0.0;
for session := 1 to session_total do
  begin
    origin := session_rqmt[session,1];
    channel := route[origin,session];
    pkt_distribution(pkt_length, session);
    pkt_time := pkt_length / capacity[channel];
    if max_pkt_time <= pkt_time then max_pkt_time := pkt_time;
    generate_event(arrival, 0.0, 0.0, pkt_time, origin, session);
  end;

if plot_switch then plot_initialize;

repeat
  current_event := base^.fptr;
  with current_event^ do
    case kind of
      arrival:
        begin
          num_of_arv[session_no] := num_of_arv[session_no] + 1;
          generate_event(arrival, time, arrival_time, pkt_time, node_no,
session_no);
          channel := route[node_no,session_no];
          if back_log[channel] = 0 then
            begin
              wait_time[channel] := 0.0;
              last_arrival_time[channel] := arrival_time;
              back_log[channel] := back_log[channel] + 1;
            end
          else
            begin
              wait_time[channel] := wait_time[channel] + pkt_time
                - ( arrival_time - last_arrival_time[channel] );
              last_arrival_time[channel] := arrival_time;
              if wait_time[channel] <= 0.0 then
                wait_time[channel] := 0.0;
              back_log[channel] := back_log[channel] + 1;
            end
          end;
        end;
    end;
  end;
end;

```

```

        generate_event(rcv_pkt, time, arrival_time, pkt_time, node_no,
session_no);
        if ((num_of_arv[plot_session] mod dot_display_freq = 0)
            and (not plot_switch)) then write('.');
        end;
    departure:
        begin
            num_of_depart[session_no] := num_of_depart[session_no] + 1;
            service_time := current_event^.time - current_event^.arrival_time
            average_delay[session_no] := average_delay[session_no] + service_
time;
            variance_delay[session_no] := variance_delay[session_no]
                + service_time * service_time;
            inter_departure_time[session_no] := current_event^.time
                - last_departure_time[session_no];
            last_departure_time[session_no] := current_event^.time;

            if (num_of_depart[session_no] mod plot_interval) = 0 then
                begin
                    if plot_session = session_no then
                        begin
                            writeln(delay_file, current_event^.time:10:3,
                                average_delay[session_no]/num_of_depart[session_no]:10:5)
                            writeln(throughput_file, current_event^.time:10:3,
                                num_of_depart[session_no]/num_of_arv[session_no]:10:5);
                            writeln(text_file, current_event^.time:10:3,
                                average_delay[session_no]/num_of_depart[session_no]:20:5,
                                num_of_depart[session_no]/num_of_arv[session_no]:18:5);
                        end;
                    if plot_switch then
                        begin
                            reset(plot_file, plot_disk_file);
                            case plot of
                                delay_curve: if plot_session = session_no then
                                    writeln(plot_file, current_event^.time:10:3,
                                        average_delay[session_no]/num_of_depart[session_no]:10:5);
                                throughput: if plot_session = session_no then
                                    writeln(plot_file, current_event^.time:10:3,
                                        num_of_depart[session_no]/num_of_arv[session_no]:10:5);
                            end; { end case }
                            close(plot_file);
                            plot_graph(plot_disk_file);
                        end;
                    end;
                end;
            end;

    rcv_pkt:
        begin
            channel := route[node_no, session_no];
            back_log[channel] := back_log[channel] - 1;

            if (num_of_depart[session_no] mod plot_interval) = 0 then

```

```

begin
  if plot_channel = channel then
    writeln(queue_file, current_event^.time:10:5,
    back_log[channel]:5);

    if plot_switch then
      begin
        reset(plot_file, plot_disk_file);
        if plot = queue_size then
          if plot_channel = channel then
            writeln(plot_file, current_event^.time:10:5,
            back_log[channel]:5);
          close(plot_file);
          plot_graph(plot_disk_file);
        end;
      end;

    node := link_matrix[channel,2];
    if node = session_rqmt[session_no,2]
    then
      { final destination }
      begin
        generate_event(departure, time, arrival_time, pkt_time, node,
session_no)
      end
    else
      { route to next node }
      begin
        channel := route[node,session_no];
        if back_log[channel] = 0 then
          begin
            wait_time[channel] := 0.0;
            back_log[channel] := back_log[channel] + 1;
            last_arrival_time[channel] := time;
          end
        else
          begin
            wait_time[channel] := wait_time[channel] + pkt_time
              - ( time - last_arrival_time[channel] );
            last_arrival_time[channel] := time;
            if wait_time[channel] <= 0.0 then
              wait_time[channel] := 0.0;
            back_log[channel] := back_log[channel] +1;
          end;
          generate_event(rcv_pkt, time, arrival_time, pkt_time, node,
session_no);
        end;
      end;
    end;
    base^.fptr := current_event^.fptr;
    current_event^.fptr^.bptr := base;
    dispose(current_event);
    until ((base^.fptr^.time >= max_time)
      or (back_log[plot_channel] >= queue_max));

```

```

if plot_switch then graph_close;

close(delay_file);
close(throughput_file);
close(queue_file);

{ collect statistics }
writeln(text_file);
writeln(text_file, 'Final Statistics:');
writeln(text_file, '=====');
writeln(text_file, ' maximum simulation time: ', max_time:10:2, ' seconds');
for session := 1 to session_total do
  writeln(text_file, ' average arrival rate of session', session:5, ' is',
    traffic_rqmt[session]:10:5, ' packets per second');
  writeln(text_file, '          number of          delay
(seconds)');
  writeln(text_file, ' session arrival departure average
variance');
  for session := 1 to session_total do
    begin
      if num_of_depart[session] = 0
      then average_delay[session] := 0.0
      else average_delay[session] := average_delay[session]
        / num_of_depart[session];

      if num_of_depart[session] <= 1
      then variance_delay[session] := 0.0
      else variance_delay[session] := (variance_delay[session]
- num_of_depart[session] * average_delay[session] * average_delay[ses-
sion])
        / (num_of_depart[session] - 1);

      writeln(text_file, session:10,
        num_of_arv[session]:10, num_of_depart[session]:10,
        average_delay[session]:15:5, variance_delay[session]:15:5);
    end;
  close(text_file);
end.

```