```
   Help
#include "optype.h"
#include "var.h"
#include "random.h"

extern char **error_msg;
extern Generator Random[];

#ifndef _WIN32

int Spawnlp( int mode, const char *cmdname, con
    st char *arg0, const char *arg1, const char *arg2
    )
{
  char cmd[MAX_PATH_LEN]="";
  int test;

  if ((strlen(cmdname)+strlen( " ")+strlen(arg1)
    +strlen(" &"))>=MAX_PATH_LEN)
  {
    Fprintf(TOSCREEN,"%s{n",error_msg[PATH_TOO_
    LONG]);
        exit(WRONG);
  }


  strcpy(cmd,cmdname);
  strcat(cmd, " ");
  strcat(cmd,arg1);
  if (mode == 1)
    strcat(cmd," &");
  Fprintf(TOSCREEN,"Opening %s {n",arg1);
  test=system(cmd);
  if((test == -1) || (test == 127))
    Fprintf(TOSCREEN, "WARNING: NO HELP AVAILABL
    E ON YOUR OS{n");
  return OK;
}
#endif

/*_____________________________OUTPUT_FILE________
```

```
    ________________________*/
extern FILE* out_stream;
extern char premiasrcdir[256];/*defined in premia
    .c*/
extern char* path_sep;/*defined in premia.c*/

/*_____________________________________CUSTOM FP
    RINTF___________________*/
int Fprintf(int user,const char s[],...)
{
  va_list ap;
  int return_value=OK;
  FILE *out;

  va_start(ap,s);

  switch (user)
    {
    case TOSCREEN:
      out=stdout;
      return_value=vfprintf(out,s,ap);
      va_end(ap);
      break;
    case TOFILE:
      out=out_stream;
      return_value=vfprintf(out,s,ap);
      va_end(ap);
      break;
    case TOSCREENANDFILE:
      out=out_stream;
      return_value=vfprintf(out,s,ap);
      va_end(ap);
      va_start(ap,s);
      out=stdout;
      return_value+=vfprintf(out,s,ap);
      va_end(ap);
      break;
    default:
      break;
    }
```

```
  return return_value;
}

/*____________________________VALIDATION__________
    ___________________*/

int Valid (int user,int status,char *helpfile)
{
  char msg,answer;
  char fhelp[MAX_PATH_LEN]="";
  int i;

  for(i=0;i<(int)strlen(helpfile);i++)
    helpfile[i]= (char)tolower(helpfile[i]);

  if ((strlen(premiasrcdir))>=MAX_PATH_LEN)
   {
    Fprintf(TOSCREEN,"%s{n",error_msg[PATH_TOO_
    LONG]);
        exit(WRONG);
   }

  /*strcpy(fhelp,premiasrcdir);
  for(i=strlen(premiasrcdir);i<(int)(strlen(helpf
    ile)+strlen(premiasrcdir));i++)
  fhelp[i] = (char)tolower(helpfile[i]);*/
  strcpy(fhelp,"..");
  strcat(fhelp,path_sep);
  strcat(fhelp,"Src");
  strcat(fhelp,helpfile);

  switch(user)
    {
    case TOSCREEN:

      if (status!=OK)
  {
    Fprintf(TOSCREEN,"{nPlease correct.{n");
  }
      else
  {
```

```
    do
      {

        Fprintf(TOSCREEN,"{nAll Right (ok: Retu
    rn, no: n, h for Help) ? {t");
        msg = (char)tolower (fgetc (stdin));
        answer = msg;
        if (answer=='h'){
#ifdef _WIN32
    _spawnlp(_P_NOWAIT,"AcroRd32.exe","_spawnl
    p",fhelp,NULL);
#endif
#ifndef _WIN32
    _spawnlp(1,"acroread","_spawnlp",fhelp,NUL
    L);
#endif
        }

                                  /* Discard rest
    of input line. */
        while (msg != '{n' && msg != EOF)
    msg = (char)fgetc (stdin);
      }
    while (answer=='h');

    status=!(answer=='{n');
  }

      Fprintf(TOSCREEN,"{n");
      break;

    case (TOSCREENANDFILE||TOFILE||NO_PAR):

      break;

    default:
      break;
    }

  return status;
}
```

```
/*_____________________________VAR SYSTEM_________
    ____________________*/

static char **formatV;
int         *true_typeV;
static char **error_msgV;

int InitVar(void)
/***********************************************
    ****************************
***
Allocates and Initializes formatV, true_typeV,
    error_msgV.
Return:
-OK if
formatV, true_typeV, error_msgV are well allocat
    ed.
-WRONG otherwise.
***********************************************
    ****************************
**/
{
  formatV=(char **)malloc(sizeof(Label)*MAX_TYPE
    );
  if (formatV==NULL)
    return 1;

  true_typeV=(int *)malloc(sizeof(int)*MAX_TYPE)
    ;
  if (true_typeV==NULL)
    return 1;

  error_msgV=(char **)malloc(sizeof(Label)*MAX_
    TYPE);
  if (error_msgV==NULL)
    return 1;

  /*For completion*/
  formatV[END]="%d";
```

```
true_typeV[END]=INT;
error_msgV[END]="Should never be asked !";

formatV[INT]="%d";
true_typeV[INT]=INT;
error_msgV[INT]="Should be an integer !";

formatV[DOUBLE]="%lf";
true_typeV[DOUBLE]=DOUBLE;
error_msgV[DOUBLE]="Should be a double !";

formatV[LONG]="%lu";
true_typeV[LONG]=LONG;
error_msgV[LONG]="Should be a long !";

formatV[PDOUBLE]="%lf";
true_typeV[PDOUBLE]=DOUBLE;
error_msgV[PDOUBLE]="Should be greater than 0!
  ";

formatV[DATE]="%lf";
true_typeV[DATE]=DOUBLE;
error_msgV[DATE]="Should be a date!";

formatV[RGDOUBLE]="%lf";
true_typeV[RGDOUBLE]=DOUBLE;
error_msgV[RGDOUBLE]="Should range betwween 0
  and 1 !";

formatV[RGDOUBLEM11]="%lf";
true_typeV[RGDOUBLEM11]=DOUBLE;
error_msgV[RGDOUBLEM11]="Should range betwwee
  n -1 and 1 !";

formatV[RGDOUBLE12]="%lf";
true_typeV[RGDOUBLE12]=DOUBLE;
error_msgV[RGDOUBLE12]="Should range betwween
  1 and 2 !";

formatV[BOOL]="%d";
true_typeV[BOOL]=INT;
```

```
error_msgV[BOOL]="Should be a Bool!";

formatV[PADE]="%d";
true_typeV[PADE]=INT;
error_msgV[PADE]="Should be a Pade!";

formatV[INT2]="%d";
true_typeV[INT2]=INT;
error_msgV[INT2]="Should be an integer great
  er than 2 !";

formatV[RGINT13]="%d";
true_typeV[RGINT13]=INT;
error_msgV[RGINT13]="Should be an integer betw
  een 1 and 3 !";

formatV[GENER]="%d";
true_typeV[GENER]=INT;
error_msgV[GENER]="Should be an integer betwee
  n 0 and 11!";

formatV[SPDOUBLE]="%lf";
true_typeV[SPDOUBLE]=DOUBLE;
error_msgV[SPDOUBLE]="Should be strictly great
  er than 0!";

formatV[RGDOUBLE051]="%lf";
true_typeV[RGDOUBLE051]=DOUBLE;
error_msgV[RGDOUBLE051]="Should range betwwee
  n 0.5 and 1 !";

formatV[DOUBLEARRAY]="%lf";
true_typeV[DOUBLEARRAY]=DOUBLE;
error_msgV[DOUBLEARRAY]="Should be an array of
   double !";

formatV[RGDOUBLE14]="%lf";
true_typeV[RGDOUBLE14]=DOUBLE;
error_msgV[RGDOUBLE14]="Should range betwween
  1 and 4 !";
```

```
  return OK;
}


int ChkVar(Planning *pt_plan, VAR *x)
/***********************************************
    ******************************
.Implements the Vtype range tests.
.Displays TOSCREEN the error message in
    error_msgV if necessary

  .pt_plan is of no use, except as an argument of
     PrintVar(pt_plan, TOSCREEN,
  x): that is in case x->Viter>=0,
  ie *x has been selected, the check is performed
     on the current value of x.

  Return:
  -OK if *x is in the range.
  -WRONG otherwise.
***********************************************
    ****************************/
{
  int status=OK;


  if (x->Vtype<FIRSTLEVEL)
    {


      switch(x->Vtype)
  {
  case END:
    Fprintf(TOSCREEN,"WARNING: CHKVAR OF END TY
    PE VAR{n");
    break;

  case BOOL:
    break;

  case PDOUBLE:
```

```
  status=(x->Val.V_PDOUBLE<0.); /* PDOUBLE>=0.
  */
  break;

case RGDOUBLE:
  status=((x->Val.V_RGDOUBLE<0.) ||(x->Val.V_
  RGDOUBLE>1.)); /*0.<=RGDOUBLE<=1.*/
  break;

case RGDOUBLEM11:
  status=((x->Val.V_RGDOUBLE<-1.) ||(x->Val.V_
  RGDOUBLE>1.)); /*-1.<=RGDOUBLE_M11<=1.*/
  break;

case RGDOUBLE12:
  status=((x->Val.V_RGDOUBLE12<1.) ||(x->Val.
  V_RGDOUBLE12>2.)); /*1.<=RGDOUBLE12<=2.*/
  break;

case INT2:
  status=(x->Val.V_INT2<2); /* 2<=INT2*/
  break;

case RGINT13:
  status=((x->Val.V_RGINT13<1) ||(x->Val.V_RG
  INT13>3)); /*1<=RGINT13<=3*/
  break;

case SPDOUBLE:
  status=(x->Val.V_PDOUBLE<=0.); /* SPDOUBLE>0
  .*/
  break;

  case RGDOUBLE051:
  status=((x->Val.V_RGDOUBLE051<0.5) ||(x->Val
  .V_RGDOUBLE051>1.)); /*0.5<=RGDOUBLE051<=1.*/
break;

case RGDOUBLE14:
  status=((x->Val.V_RGDOUBLE14<1.) ||(x->Val.
  V_RGDOUBLE14>4.)); /*1.<=RGDOUBLE12<=4.*/
```

```
  break;

case GENER:
  status=((x->Val.V_GENER<0)||(x->Val.V_GENER>
  GEN_NUMBER)); /* V_GENER>0.*/
  break;

default:
  break;
}

    if (status!=OK)
{
  Fprintf(TOSCREEN,"{nBad value:{n");
  PrintVar(pt_plan,TOSCREEN,x);
  Fprintf(TOSCREEN,"%s",error_msgV[x->Vtype]);
}

  }

else
  {

    switch(x->Vtype)
{
case (NUMFUNC_1):
  status=ChkParVar(pt_plan,(x->Val.V_NUMFUNC_1
  )->Par);
  break;

case (NUMFUNC_2):
  status=ChkParVar(pt_plan,(x->Val.V_NUMFUNC_2
  )->Par);
  break;

case (PTVAR):
  status=ChkParVar(pt_plan,(x->Val.V_PTVAR)->
  Par);
  break;

default:
```

```
    break;
  }

    }

  return status;
}


void ExitVar(void)
     /*****************************************
    *********************************
***
Desallocates formatV, true_typeV, error_msgV.
************************************************
    ****************************
**/
{
  free(formatV);
  free(true_typeV);
  free(error_msgV);

  return;
}


int FprintfVar(int user,const char s[],VAR *x)
{
  int vt=true_typeV[x->Vtype],return_value=0;

  switch(vt)
    {
    case DOUBLE:

      return_value=Fprintf(user,s,x->Val.V_
    DOUBLE);
      break;

    case INT:

      return_value=Fprintf(user,s,x->Val.V_INT);
```

```
      break;

    case LONG:

      return_value=Fprintf(user,s,x->Val.V_LONG);
      break;

    default:
      Fprintf(TOSCREEN,"WARNING: UNKNOWN TRUETYPE
     IN THE VAR SYSTEM{n");
      return_value=0;
      break;
    }

  return return_value;
}


int PrintVar(Planning *pt_plan,int user,VAR *x)
     /*****************************************
    *********************************
***
Print the name and/or the value of *x depending
    on x->Viter and user:

 .PrintVar(&plan,NAMEONLYTOFILE,x):
 Fprint(TOFILE,"#%s{n",x->Vname);

  .PrintVar(&plan,VALUEONLYTOFILE,x):
  Fprint(TOFILE,"formatV[x->Vtype]{t",x->Vname);

    .PrintVar(&plan,TOFILE,x):
    if x->Viter==ALLOW or FORBID, (ie *x has no
    t been selected for iteration)
    Fprint(TOFILE,"#%s{tformatV[x->Vtype]{n,x->
    Vname,x->Val);
    else
    Fprint(TOFILE,"#%s{t:from formatV[x->Vtype]
    to formatV[x->Vtype] step
    %d{n",
    x->Vname,Min.Val,Max.Val,StepNumber);
```

```
    where Min, Max and StepNumber are the fields
     of plan->Par[Viter].

     PrintVar(&plan,TOSCREENANDFILE,x): the sa
    me with
     Fprintf(TOSCREENANDFILE,..)

    .PrintVar(&plan,TOSCREEN,x):
    if x->Viter==ALLOW or FORBID, (ie *x has no
    t been selected for iteration)
    Fprint(TOSCREEN,"%s{tformatV[x->Vtype]{n,x-
    >Vname,x->Val);
    else
    Fprint(TOSCREEN,"%s{t:from formatV[x->Vtype
    ] to formatV[x->Vtype] step
    %d{n",
    x->Vname,Min.Val,Max.Val,StepNumber);
    where Min, Max and StepNumber are the fiel
    ds of plan->Par[Viter].

     !!!!!!!!!!!!!! WARNING!!!!!!!!!!!!!!
     (i) Fprintf(user,"formatV[x->Vtype] format
    V[y->Vtype]",x->Val,y->Val)
     DOES NOT WORK,
     so such a Fprintf is cut into parts.
     (ii) No test is made to check the temporar
    y string: char
     string[MAX_CHAR]; is smaller than MAX_CHAR
     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!

      Return:
      -the return_value of the last Fprintf
*************************************************
    ******************************/
{
  char string[MAX_CHAR_X4];
  Iterator* pt_it;
  int return_value=1;

  if (x->Vtype<FIRSTLEVEL)
```

```
    {

      if (x->Viter!=IRRELEVANT)
  {
    switch (user)
      {
      case TOSCREEN:

        if (x->Viter >= ALREADYITERATED)
    {

      pt_it=&(pt_plan->Par[x->Viter-ALREADYIT
    ERATED]);

      strcpy( string, x->Vname );
      strcat( string, " from " );
      strcat(string,formatV[x->Vtype]);
      FprintfVar(TOSCREEN,string, &(pt_it->Min)
    );
      strcpy( string, " to " );
      strcat(string,formatV[x->Vtype]);
      FprintfVar(TOSCREEN,string, &(pt_it->Max)
    );
      strcpy(string," step %d");
      strcat(string,"{n");

      return_value=Fprintf(TOSCREEN,string,pt_
    it->StepNumber);
    }
        else if ((x->Viter==ALLOW)||(x->Viter==
    FORBID))
    {
      if ((x->Vtype==GENER) && (0<=x->Val.V_GEN
    ER)&&(x->Val.V_GENER<GEN_NUMBER))
       {
          strcpy(string, x->Vname );
        strcat(string, ":{t" );
        strcat(string,formatV[x->Vtype]);
        strcat(string," (");
        strcat(string,Random[x->Val.V_GENER].
    Name);
```

```
      strcat(string," )");

     } else
       {
       /*printf("%d{n",x->Vtype);
         printf("%s{n",x->Vname);*/

       strcpy(string, x->Vname );
       strcat(string, ":{t" );
       strcat(string,formatV[x->Vtype]);
       /*printf("%d{n",x->Vtype);*/
       }
   strcat(string, "{n" );
   return_value=FprintfVar(TOSCREEN,string,
x);
}
     else
{
   pt_it=&(pt_plan->Par[x->Viter]);

   strcpy(string, x->Vname );
   strcat( string, " from " );
   strcat(string,formatV[x->Vtype]);
   FprintfVar(TOSCREEN,string, &(pt_it->Min)
);
   strcpy( string, " to " );
   strcat(string,formatV[x->Vtype]);
   FprintfVar(TOSCREEN,string, &(pt_it->Max)
);
   strcpy(string," step %d");
   strcat(string,"{n");

   return_value=Fprintf(TOSCREEN,string,pt_
it->StepNumber);
}
     break;

   case TOFILE:

     if (x->Viter >= ALREADYITERATED)
{
```

```
  pt_it=&(pt_plan->Par[x->Viter-ALREADYIT
ERATED]);

  strcpy(string,"#");
  strcat( string, x->Vname );
  strcat( string, " from " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOFILE,string, &(pt_it->Min));
  strcpy( string, " to " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOFILE,string, &(pt_it->Max));
  strcpy(string," step %d");
  strcat(string,"{n");

  return_value=Fprintf(TOFILE,string,pt_it-
>StepNumber);
}
    else if ((x->Viter==ALLOW)||(x->Viter==
FORBID))
{
  strcpy(string,"#");
  strcat( string, x->Vname );
  strcat( string, ":{t" );
  strcat(string,formatV[x->Vtype]);
  strcat( string, "{n" );

  return_value=FprintfVar(TOFILE,string, x)
;
}
    else
{
  pt_it=&(pt_plan->Par[x->Viter]);

  strcpy(string,"#");
  strcat(string, x->Vname );
  strcat(string, " from " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOFILE,string, &(pt_it->Min));
  strcpy(string, " to " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOFILE,string, &(pt_it->Max));
```

```
  strcpy(string," step %d");
  strcat(string,"{n");

  return_value=Fprintf(TOFILE,string,pt_it-
>StepNumber);
}
    break;

  case TOSCREENANDFILE:

    if (x->Viter >= ALREADYITERATED)
{
  pt_it=&(pt_plan->Par[x->Viter-ALREADYIT
ERATED]);

  strcpy(string,"#");
  strcat( string, x->Vname );
  strcat( string, " from " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOSCREENANDFILE,string, &(pt_
it->Min));
  strcpy( string, " to " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOSCREENANDFILE,string, &(pt_
it->Max));
  strcpy(string," step %d");
  strcat(string,"{n");

  return_value=Fprintf(TOSCREENANDFILE,str
ing,pt_it->StepNumber);
}
    else if ((x->Viter==ALLOW)||(x->Viter==
FORBID))
{
  strcpy(string,"#");
  strcat( string, x->Vname );
  strcat( string, ":{t" );
  strcat(string,formatV[x->Vtype]);
  strcat( string, "{n" );

  return_value=FprintfVar(TOSCREENANDFILE,
```

```
string, x);
}
    else
{
  pt_it=&(pt_plan->Par[x->Viter]);

  strcpy(string,"#");
  strcat( string, x->Vname );
  strcat( string, " from " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOSCREENANDFILE,string, &(pt_
it->Min));
  strcpy( string, " to " );
  strcat(string,formatV[x->Vtype]);
  FprintfVar(TOSCREENANDFILE,string, &(pt_
it->Max));
  strcpy(string," step %d");
  strcat(string,"{n");

  return_value=Fprintf(TOSCREENANDFILE,str
ing,pt_it->StepNumber);
}
    break;


  case NAMEONLYTOFILE:

    return_value=Fprintf(TOFILE,"#%s{n",x->
Vname);
    break;

  case VALUEONLYTOFILE:

    strcpy(string,formatV[x->Vtype]);
    strcat( string, "{t" );
    return_value=FprintfVar(TOFILE,string, x
);
    break;

  default:
    break;
```

```
      }
  }
    }
  else
    {
      switch(x->Vtype)
  {
  case (NUMFUNC_1):
    return_value=ShowParVar(pt_plan,user,(x->Val
    .V_NUMFUNC_1)->Par);
    break;

  case (NUMFUNC_2):
    return_value=ShowParVar(pt_plan,user,(x->Val
    .V_NUMFUNC_2)->Par);
    break;

  case (PTVAR):
    return_value=ShowParVar(pt_plan,user,(x->Val
    .V_PTVAR)->Par);
    break;

  case (DOUBLEARRAY):
    return_value=OK;
    break;

  default:
    break;
  }
    }

  return return_value;

}



int ScanVar(Planning *pt_plan,int user,VAR *x)
{
  Iterator *pt_iterator;
  int return_value=1;
```

```
int msg,answer;

if (x->Vtype<FIRSTLEVEL)
  {

    if (x->Viter!=IRRELEVANT)
{
  /*Print the current value*/
  PrintVar(pt_plan,user,x);

  if ((pt_plan->Action=='p')&&(pt_plan->VarNu
  mber<(MAX_ITERATOR-1))&&(x->Viter!=FORBID)&&(x->
  Viter<ALREADYITERATED))

                              /*if ((pt_plan->
  VarNumber<(MAX_ITERATOR-1))&&(x->Viter==ALLOW))*/

    {
      Fprintf(TOSCREEN,"{t Ok:Return Modify:m
  Iter:i {t?");

      answer = tolower(fgetc(stdin));
      msg = answer;
      while( (answer != '{n') && (answer != EO
  F))
  answer = fgetc(stdin);

      switch(msg)
  {
  case 'i':
    if (x->Vtype==GENER)
     Display_Generators();
    if (x->Viter==ALLOW) /*x has not been se
  lected before*/
      {
        pt_iterator=&(pt_plan->Par[pt_plan->
  VarNumber]);
        pt_iterator->Location=x;
        (void)CopyVar(x,&(pt_iterator->Defau
  lt));
```

```
      x->Viter=pt_plan->VarNumber;

      pt_iterator->Min.Vtype=x->Vtype;
      pt_iterator->Max.Vtype=x->Vtype;

      pt_plan->VarNumber=pt_plan->VarNumb
er+1;
      (pt_plan->Par[pt_plan->VarNumber]).Mi
n.Vtype=END;

      strcpy(pt_iterator->Min.Vname,x->Vnam
e);
      strcpy(pt_iterator->Max.Vname,x->Vnam
e);

      pt_iterator->Min.Viter=FORBID;
      pt_iterator->Max.Viter=FORBID;
    }
  else if (x->Viter>ALREADYITERATED)
    {
      pt_iterator=&(pt_plan->Par[x->Viter-
ALREADYITERATED]);
    }
  else /*x has already been selected befor
e*/
    {
      pt_iterator=&(pt_plan->Par[x->Viter])
;
    }

  Fprintf(TOSCREEN,"Min value{t?");
  do {
    scanf(formatV[x->Vtype],&((pt_iterator-
>Min).Val.V_INT));
  /*CopyVar(&(pt_iterator->Min),x);*/

  } while (ChkVar(pt_plan,&(pt_iterator->Mi
n))!=OK);

  Fprintf(TOSCREEN,"Max value{t?");
  do {
```

```
    scanf(formatV[x->Vtype],&((pt_iterator-
>Max).Val.V_INT));
  } while
    ((ChkVar(pt_plan,&(pt_iterator->Max))!=
OK)||(LowerVar(user,&(pt_iterator->Min),&(pt_it
erator->Max))!=OK));

  Fprintf(TOSCREEN,"Number of steps{t?");
  do
  {
    return_value=scanf("%d%*c",&(pt_itera
tor->StepNumber));
  } while (ChkStepNumber(user,pt_iterator,
pt_iterator->StepNumber)!=OK);

  break;

case '{n':
  return_value=OK;
  break;

case 'm':
  if (x->Vtype==GENER)
  Display_Generators();
  if (x->Viter!=ALLOW)
    ShrinkPlanning(x->Viter,pt_plan);
  x->Viter=ALLOW;
  Fprintf(TOSCREEN,"Value{t?");
  return_value=scanf(formatV[x->Vtype],&(x-
>Val.V_INT));
  scanf("%*c");
  break;

default:
  return_value=OK;
  break;
}
  }
else

                              /*else if (x->Vi
```

```
  ter==ALLOW)*/
    {

      Fprintf(TOSCREEN,"{t Ok:Return Modify:m{
  t?");

      answer = tolower(fgetc(stdin));
      msg = answer;
      while ((answer != '{n') && (answer != EO
  F))
  answer = fgetc(stdin);

      switch(msg)
  {
  case '{n':
    return_value=OK;
    break;
  case 'm':
    Fprintf(TOSCREEN,"Value{t?");
    return_value=scanf(formatV[x->Vtype],&(x-
  >Val.V_INT));
    scanf("%*c");
    break;

  default:
    return_value=OK;
    break;
  }

    }

}/*Irrelevant*/
  }
else /*Vtype>FirstLevel*/
  {

    switch(x->Vtype)
{
case (NUMFUNC_1):
  do
    {
```

```
        return_value=GetParVar(pt_plan,user,(x->
    Val.V_NUMFUNC_1)->Par);
      }
    while (ChkParVar(pt_plan,(x->Val.V_NUMFUNC_1
    )->Par)!=OK);
    break;

  case (NUMFUNC_2):
    do
      {
        return_value=GetParVar(pt_plan,user,(x->
    Val.V_NUMFUNC_2)->Par);
      }
    while (ChkParVar(pt_plan,(x->Val.V_NUMFUNC_2
    )->Par)!=OK);
    break;


  case (PTVAR):
    return_value=GetParVar(pt_plan,user,(x->Val.
    V_PTVAR)->Par);
    break;

  default:
    break;
  }

    }

  return return_value;
}


int ChkParVar(Planning *pt_plan,VAR *x)
     /*****************************************
    ********************************
****
The list version of the former.

 Return:
 -OK if every item has a pertaining value.
```

```
 -The number of bad values otherwise.

  ********************************************
    ******************************
  **/
{
  int status=OK;

  while (x->Vtype!=END)
    {
      status+=ChkVar(pt_plan,x);
      x++;
    }

  return status;
}

int GetParVar(Planning *pt_plan,int user,VAR *x)
     /*****************************************
    *********************************
****
The list version of ScanVar.

 Return:
 -OK if every item has been well ScanVared.
 -The number of bad scans otherwise.

  ********************************************
    ******************************
  **/
{
  int status=OK;

  while (x->Vtype!=END)
    {
      if (x->Viter>ALREADYITERATED){
  x++;
      }
      else{
  status+=(ScanVar(pt_plan,user,x)<=0);
  x++;
```

```
      }
    }
  return status;
}

int ShowParVar(Planning *pt_plan,int user,VAR *x)
     /*****************************************
    ********************************
****
.The list version of PrintVar.


  Return:
  -OK if every item has been well PrintVared.
  -NO_PAR if the list is empty.
  -The number of bad print messages otherwise.
  ***********************************************
    ****************************
  ****/
{
  int status=OK;

  if (x->Vtype==END)
    return NO_PAR;

  while (x->Vtype!=END)
    {
      status+=(PrintVar(pt_plan,user,x)<0);
      x++;
    }

  return status;
}

int LowerVar(int user,VAR *x, VAR*y)
     /*****************************************
    ********************************
****
.Displays to user a message if x->Val>y->Val.

 !!!!!!!!!!!!!! WARNING!!!!!!!!!!!!!!
```

```
 Assumes that x->Vtype==y->Vtype, no check is pe
    rformed;
 the Vtype is read in x->Vtype
 AND is assumed to be in the range of the beginn
    ing ifs; otherwise
 the return value is OK
 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!


    Return:
    -OK if x->Val<=y->Val, cf also WARNING
    -WRONG otherwise.

     ****************************************
    **********************************
     **/
{
  int status=OK;
  int vt=true_typeV[x->Vtype];

  switch(vt)
    {
    case DOUBLE:

      status+=(x->Val.V_DOUBLE>y->Val.V_DOUBLE);
      break;

    case INT:

      status+=(x->Val.V_INT>y->Val.V_INT);
      break;

    case LONG:

      status+=(x->Val.V_LONG>y->Val.V_LONG);
      break;

    default:
      break;
    }
```

```
  if (status!=OK)
    Fprintf(user,"Min %s should be less than Max
    %s{n",x->Vname,y->Vname);

  return status;
}


void CopyVar(VAR *srce,VAR *dest)
{
  memcpy(dest,srce,sizeof(VAR));
}



/*_____________________________________PLANNING__
    _______________________________*/
void ResetPlanning(Planning *pt_plan)
     /*****************************************
    *********************************
****
.Reset *pt_plan:
.at the first call, (pt_plan->Par)[0] is set to
    END, pt_plan->VarNumber to
0.
.the next calls in addition (and before) the pt_
    plan->Par[i].Viter are set to
ALLOW
************************************************
    ***************************
***/
{
  static int first=1;
  int i;

  if (first)
    {
      first=0;
    }
  else
```

```
    {
      for (i=0;i<pt_plan->VarNumber;i++){
  pt_plan->Par[i].Default.Viter=ALLOW;
  (void)CopyVar(&(pt_plan->Par[i].Default),pt_pl
    an->Par[i].Location);
      /*  (pt_plan->Par[i].Location)->Viter=AL
    LOW;  */
      }
    }

  (pt_plan->Par)[0].Min.Vtype=END;
  pt_plan->VarNumber=0;
  pt_plan->NumberOfMethods=0;

  return;
}

void ShowPlanning(int user,Planning *pt_plan)
     /*****************************************
    *********************************
****
.Displays the iterated VARs selected in *pt_plan,
     preceded by {n##{n and
followed by ##{n
in case user==NAMEONLYTOFILE

 .If user is not NAMEONLYTOFILE, VALUEONLYTOFILE
    or TOSCREEN,
 doesn't do anything.

  ***********************************************
    ******************************
  **/
{
  int i;

  switch(user)
    {
    case NAMEONLYTOFILE:

      Fprintf(TOFILE,"{n##{n");
```

```
      for (i=1;i<=pt_plan->VarNumber;i++)
  PrintVar(pt_plan,NAMEONLYTOFILE,(pt_plan->Par)
    [i-1].Location);
      Fprintf(TOFILE,"##{n");

      break;
    case VALUEONLYTOFILE:

      for (i=1;i<=pt_plan->VarNumber;i++)
  PrintVar(pt_plan,VALUEONLYTOFILE,(pt_plan->
    Par)[i-1].Location);

      break;

    case TOSCREEN:

      for (i=1;i<=pt_plan->VarNumber;i++)
  PrintVar(pt_plan,TOSCREEN,(pt_plan->Par)[i-1].
    Location);

      break;
    default:
      break;
    }

  return;
}

void ShrinkPlanning(int index,Planning *pt_plan)
     /*****************************************
    *********************************
****
.Removes the item no index in *pt_plan and shrink
    s *pt_plan, resetting the
coreesponding values
of the Viter fields of the selected VARs.
************************************************
    ***************************
***/
{
  int i;
```

```
  Iterator *pt_it, *pt_next_it;

  pt_it=&(pt_plan->Par[index]);


  for (i=index;i<pt_plan->VarNumber;i++)
    {
      pt_next_it=&(pt_plan->Par[i+1]);

      (void)CopyVar(&(pt_next_it->Min),&(pt_it->
    Min));
      (void)CopyVar(&(pt_next_it->Max),&(pt_it->
    Max));
      (void)CopyVar(&(pt_next_it->Default),&(pt_
    it->Default));
      pt_it->Location=pt_next_it->Location;

      if (pt_it->Min.Vtype!=END)
  (pt_it->Location)->Viter-=1;

      pt_it=pt_next_it;
    }

  pt_plan->VarNumber-=1;

  /*pt_plan->Par[pt_plan->VarNumber].Min.Vtype=EN
    D;*/

  return;
}

int ChkStepNumber(int user,Iterator *pt_iterator,
    int step)
     /*****************************************
    ********************************
****
.Checks for step to be in the range [1,MAX_ITER]
    which is defined in
optype.h.
.Displays to user an error message if not.
```

```
 Return:
 -OK if step is in the range.
 -WRONG otherwise.

  ********************************************
    ******************************
  **/
{
  static int INT_dummy;
  static long LONG_dummy;

  int vt=true_typeV[pt_iterator->Min.Vtype];

  if ((step<1)|| (step>MAX_ITER))
    {
      Fprintf(user,"should range between 1 and %
    d{n",MAX_ITER);
      return WRONG;
    }

  switch(vt)
    {
    case INT:
      INT_dummy=(pt_iterator->Max.Val.V_INT-pt_it
    erator->Min.Val.V_INT)/(int)(pt_iterator->StepNumb
    er);
      if (INT_dummy<1)
  {
    pt_iterator->StepNumber=pt_iterator->Max.Val
    .V_INT-pt_iterator->Min.Val.V_INT;
    Fprintf(user,"WARNING: NUMBER OF STEPS SET
    TO %d{n",pt_iterator->StepNumber);
  }
      break;

    case LONG:
      LONG_dummy=(pt_iterator->Max.Val.V_LONG-pt_
    iterator->Min.Val.V_LONG)/(long)(pt_iterator->Ste
    pNumber);
      if (LONG_dummy<1)
  {
```

```
    pt_iterator->StepNumber=(int)(pt_iterator->
    Max.Val.V_LONG-pt_iterator->Min.Val.V_LONG);
    Fprintf(user,"WARNING: NUMBER OF STEPS SET
    TO %d{n",pt_iterator->StepNumber);
  }

      break;

    default:
      break;
    }


  return OK;
}


void NextValue(int count,Iterator* pt_iterator)
/***********************************************
    ****************************
.Compute the next value of pt_iterator.Location->
    Val according to the fields
of pt_iterator

 !!!!!!!!!!!!!! WARNING!!!!!!!!!!!!!!
 Assumes that pt_iterator->Min.Vtype is in the
    range of the beginning ifs;
 otherwise
 dosn't do anything
 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!

***********************************************
    ******************************/
{
  static double DOUBLE_dummy;
  static int INT_dummy;
  static long LONG_dummy;
  int vt=true_typeV[pt_iterator->Min.Vtype];

  /*FprintfVar(TOSCREEN,formatV[(pt_iterator->Loc
```

```
  ation)->Vtype],pt_iterator->Location);*/

switch(vt)
  {
  case DOUBLE:
    DOUBLE_dummy=(pt_iterator->Max.Val.V_PDOUBL
  E-pt_iterator->Min.Val.V_PDOUBLE)/(double)(pt_it
  erator->StepNumber);
    (pt_iterator->Location)->Val.V_PDOUBLE+=
  DOUBLE_dummy;
    if (count==(pt_iterator->StepNumber-1))
  (pt_iterator->Location)->Val.V_PDOUBLE=pt_
  iterator->Max.Val.V_PDOUBLE;
    break;

  case INT:
    INT_dummy=(pt_iterator->Max.Val.V_INT-pt_it
  erator->Min.Val.V_INT)/(int)(pt_iterator->StepNumb
  er);
    (pt_iterator->Location)->Val.V_INT+=INT_dum
  my;
    if (count==(pt_iterator->StepNumber-1))
(pt_iterator->Location)->Val.V_INT=pt_iterator
  ->Max.Val.V_INT;
    break;

  case LONG:
    LONG_dummy=(pt_iterator->Max.Val.V_LONG-pt_
  iterator->Min.Val.V_LONG)/(long)(pt_iterator->Ste
  pNumber);
    (pt_iterator->Location)->Val.V_LONG+=LONG_
  dummy;
    if (count==(pt_iterator->StepNumber-1))
(pt_iterator->Location)->Val.V_LONG=pt_itera
  tor->Max.Val.V_LONG;
    break;

  default:
    break;
  }
```

```
  return;
}

int ShowParVarTestRes(Planning *pt_plan, int use
    r, VAR *x)

     /* This routine displays needed values and
    needed arrays to plot stock
and P&L trajectories IN COLUMNS in the premia.ou
    t file

  BE CAREFULL : doublearrays must be the LAST arg
    uments of Test->Res[] !!!!

        if you want to put a new output argument
    in Test->Res[], you must
        right-shift the indices of the arrays
        */

{
  int status=OK, k;
  long size;
  VAR *y;
  char string[MAX_CHAR_X3];

  if (x->Vtype==END)
    return NO_PAR;

  while ((x->Vtype!=END)&&(x->Vtype!=DOUBLEARRAY)
    )
    {
      status+=(PrintVar(pt_plan,user,x)<0);
      x++;
    }


  if (x->Vtype!=END)
    {
      size=x->Val.V_DOUBLEARRAY->size;
      for (k=0;k<size;k++)
  {
```

```
    y=x;
    while (y->Vtype!=END)
      {
        if (y->Val.V_DOUBLEARRAY->size==2)
    {
      strcpy(string,formatV[y->Vtype]);
      strcat(string," ");
      strcat(string,formatV[y->Vtype]);
      strcat(string," ");
      status+=(Fprintf(user,string,y->Val.V_
    DOUBLEARRAY->array[0],y->Val.V_DOUBLEARRAY->array[1])<
    0);
      y++;
    }
        else if (y->Val.V_DOUBLEARRAY->size>k)
    {
      strcpy(string,formatV[y->Vtype]);
      strcat(string," ");
      status+=(Fprintf(user,string,y->Val.V_
    DOUBLEARRAY->array[k])<0);
      y++;
    }
      else
        y++;
      }
    Fprintf(user,"{n");
  }
    }


  return status;
}
```

References