```
   Help
#include "optype.h"

#include "var.h"

#include "method.h"

#include "error_msg.h"


extern char **error_msg;

extern char *path_sep;


/*_____________________________________PRICING
    METHODS_____________________________________*/

int   GetMethod(int user,Planning *pt_plan,Pric
    ing *Pr,PricingMethod *Met)

{

  (Met->Init)(Met);



  if (user==TOSCREEN)

    {

      if (ShowMethod(user,pt_plan,Pr,Met))

  {

    do

      {

        Fprintf(TOSCREEN,"%s{n",Met->Name);
```

```
        if (pt_plan->NumberOfMethods != 0)

    FixMethod(pt_plan,Met);

        GetParVar(pt_plan,user,Met->Par);

      }

    while (ShowMethod(user,pt_plan,Pr,Met));

  }

    }


  return ShowMethod(TOSCREENANDFILE,pt_plan,Pr,
    Met);

}


int ShowMethod(int user,Planning *pt_plan,Pricing
     *Pr,PricingMethod *Met)

{

  char helpfile[MAX_PATH_LEN]="";

  int pos;

  char *pdest;

  /*   char *par[10]; */

  (Met->Init)(Met);

  if ((2*strlen(Pr->ID)+4*strlen(path_sep)+strle
    n("mod") +strlen(Met->Name)
    +strlen("_doc.pdf"))
```

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


strcpy(helpfile,path_sep);

strcat(helpfile,"mod");

strcat(helpfile,path_sep);



pdest=strchr(Pr->ID,'_');

pos=pdest-Pr->ID;



strncat(helpfile,Pr->ID,pos);



strcat(helpfile,path_sep);



strcat(helpfile,Pr->ID);

strcat(helpfile,path_sep);

strcat(helpfile,Met->Name);

strcat(helpfile,"_doc.pdf");



/*printf("%s{n",helpfile);*/
```

```
if (user==TOSCREEN)

  FixMethod(pt_plan,Met);


if (user==TOSCREENANDFILE)

  {

    Fprintf(user,"##PricingMethod:%s{n",Met->Na
  me);

    ShowParVar(pt_plan,user,Met->Par);

  }

else

  {

    if (ShowParVar(pt_plan,user,Met->Par)==OK)

{

  return Valid(user,(Met->Check)(user, pt_plan
  , Met)+ChkParVar(pt_plan,Met->Par),helpfile);

}

    else

return Valid(NO_PAR,ChkParVar(pt_plan,Met->
  Par),helpfile);

  }
```

```
  return OK;

}



int ShowResultMethod(int user,Planning *pt_plan,
    int error, PricingMethod *Met)

{



  if ((error==0)||(user==NAMEONLYTOFILE))

    {

      ShowParVar(pt_plan,user,Met->Res);

    }

  else
    {
      Fprintf(user,"%s{n",error_msg[error]);

    }



  return OK;

}



int FixMethod(Planning *pt_plan, PricingMethod *
    pt_method)

{

  int i,j;
```

```
char msg,answer;



if (pt_plan->NumberOfMethods == 0)

  return OK;



for (j=0; j<pt_plan->VarNumber && pt_plan->Par[
  j].Location->Vtype !=END; ++j){

  for (i=0; pt_method->Par[i].Vtype!=END; ++i){


    if (!strcmp(pt_method->Par[i].Vname,pt_plan
  ->Par[j].Location->Vname))

{

  pt_plan->Par[j].Location = &pt_method->Par[
  i];

  pt_method->Par[i].Viter=ALREADYITERATED+j;

  return DONOTITERATE;

}

    else if (CompareParameterNames(pt_method->
  Par[i].Vname,pt_plan->Par[j].Location->Vname)==OK)

{

  Fprintf(TOSCREEN,"{nWould you like to itera
  te {"%s{" like {"%s{" {n{t{t (ok: Return, no: n) ?
   {t",pt_method->Par[i].Vname, pt_plan->Par[j].
  Location->Vname);
```

```
    answer = (char)tolower(fgetc(stdin));

    msg = answer;

    while( (answer != '{n') && (answer != EOF))

      answer = (char)fgetc(stdin);



    if (msg=='{n')

      {

        pt_plan->Par[j].Location = &pt_method->
    Par[i];

        pt_method->Par[i].Viter=ALREADYITERATED+
    j;

        return DONOTITERATE;

      }

  }

      else if  (pt_method->Par[i].Vtype == pt_pl
    an->Par[j].Location->Vtype)

  {

    Fprintf(TOSCREEN,"{nWould you like to itera
    te {"%s{" like {"%s{" {n{t{t (ok: Return, no: n) ?
     {t",pt_method->Par[i].Vname, pt_plan->Par[j].
    Location->Vname);

    answer = (char)tolower(fgetc(stdin));

    msg = answer;

    while( (answer != '{n') && (answer != EOF))
```

```
      answer = (char)fgetc(stdin);


    if (msg==’{n’)
      {
        pt_plan->Par[j].Location = &pt_method->
    Par[i];
        pt_method->Par[i].Viter=ALREADYITERATED+
    j;
        return DONOTITERATE;
      }
  }
    }
  }
  return OK;
}


int CompareParameterNames(char *s1, char *s2)
{
  /*
    This functions compares 2 strings for the oc
    curance of one withing the other.

    Returns:
```

```
     OK=0     if either  string s1 is containe
  d in string s2, or if s2 is contained in s1

               (ignoring beginning '#' charac
  ters)

     WRONG=1  otherwise.

*/

int len,i;

char *small, *large;


if (strlen(s1) <= strlen(s2)) {

  small=s1;

  large=s2;

}

else {

  small=s2;

  large=s1;

}


/* strip away beginning '#' characters */

if (small[0]=='#')

  ++small;
```

```
  if (large[0]==’#’)
    ++large;

  len=strlen(large);
  for (i=0; i<= (int)(len-strlen(small)); ++i)
    {
      if (strncmp(small,large,strlen(small))==0)
  {
    return OK;
  }
      ++large;
    }
  return WRONG;
}
```

References