Error: Macro Include(TocTme) failed

```
pre-0.11 Wiki macro Include by provider <class 'includemacro.macros.IncludeMacro'> no longer sup
```

TME1 : Outils de développement de programmes C

Objectifs

L'objectif de ce premier TME est de vous permettre de vérifier que vous maîtrisez les principaux outils de développement utilisés pour programmer efficacement en langage C sous UNIX. Attention : Cette U.E. suppose que vous connaissez le langage C. Si vous ne le connaissez pas, ou si vous avez des lacunes, vous devez vous formez par vous même. Les bibliothèques sont faites pour cela, et vous pouvez utiliser les machines des salles de TP en libre service pour faire des exercices.

Dans ce TME, on utilisera principalement les outils suivants:

- vim ou emacs : éditeur de texte
- gcc : préprocesseur, compilateur, éditeur de liens
- make : génération automatique avec gestion des dépendances

L'application logicielle proposée est une application de manipulation d'images au format pgm. Il s'agit d'images de hauteur et largeur quelconques quelconque, ou chaque pixel est codé sur un octet (256 niveaux de gris). Les algorithmes réalisent fondamentalement des parcours de tableaux à 2 dimensions.

Commencez par créer un répertoire tme1, qui contiendra tous les fichiers utilisés dans ce TME.

Etape 1 : programme principal

Commencez par créer un sous-répertoire tme1/etape1 dans le répertoire tme1, et placez-vous dans ce répertoire. Dans cette première étape, vous allez devoir écrire deux fichiers : un fichier "main.c" contenant la fonction main(), et un fichier "Makefile" permettant de compiler le programme et de générer un fichier exécutable "pgmg".

Le fichier "Makefile" doit contenir deux règles :

```
pgmg : main.c
```

qui effectue la compilation ET l'édition de liens en une seule passe.

```
clean
```

qui efface les fichiers temporaires et un éventuel 'core', pour ne conserver que les fichiers sources.

Le prototype de la fonction main() est défini comme suit:

```
int main(int argc, char *argv[])
```

Dans cette première étape, le tableau argv[] contient exactement 3 arguments: nom du programme, nom du fichier d'entrée, nom du fichier de sortie.

Le programme main() lit le fichier "input_file_name", en utilisant la fonction readpgm(). Il recopie chaque octet dans un tampon que vous devez allouer. Il écrit le contenu de ce tampon dans un fichier "output_file_name", en utilisant la fonction writepgm().

Pour pouvoir utiliser les fonctions readpgm() et writepgm() dans votre programme main(), vous devez inclure le fichier "pgmio.h" dans le fichier main.c. Il se trouve dans:

```
/users/enseig/encadr/cao/include/pgmio.h
```

Les prototypes des fonctions de lecture et d'écriture du format pgm, ainsi que la structure de données permettant de représenter une image en mémoire sont définis dans le fichier "pgmio.h". Vous pouvez retrouver cette information dans la page PgmInputOutput.

Pour ce qui concerne le Makefile, l'édition de liens doit se faire avec la bibliothèque C qui se trouve dans

```
/users/enseig/encadr/cao/lib/libpgmio.a
```

Lorsque l'écriture du programme main.c et du makefile sont terminés, lancez la compilation et l'édition de lien avec la commande

```
$ make
```

Un programme exécutable pgmg doit être généré dans votre répertoire de travail. Pour vérifier que le programme fonctionne, recopiez dans votre répertoire de travail le fichier "Untitled" contenant une image au format pgm. Ce fichier se trouve dans:

```
/users/enseig/encadr/cao/tme1/Untitled.pgm
```

Ce fichier peut être visualisé en utilisant la commande

```
$ display Untitled.pgm
```

Lancez l'exécution du programme pgmg:

```
$ ./pgmg Untitled.pgm new.pgm
```

Vérifiez que le fichier de sortie contient bien l'image initiale.

Etape 2 : Analyse de la ligne de commande

Commencez par créer un second sous-répertoire tme1/etape2 dans votre répertoire tme1. Recopiez les fichiers main.c et Makefile de dans ce sous-répertoire. Dans cette seconde étape, ce n'est plus le programme main() qui analyse la ligne de commande, car on souhaite introduire des arguments optionnels dans la ligne de commande. La tâche d'analyse est effectuée par une fonction getarg() contenue dans le fichier "getarg.c". Il faut donc modifier la fonction main(), et écrire la fonction getarg().

Le fichier "getarg.c" doit contenir aussi la déclaration de la variable globale mainarg, de type 'mainarg_t', qui contient les valeurs de tous les paramètres lus par la fonction getarg().

Le fichier "getarg.h" doit contenir la définition du type mainarg_t, et la déclaration du prototype de la fonction getarg() :

```
typedef struct mainarg_t {
char     *InputFileName;
char     *OutputFileName;
char     *CommandFileName
char     verbose;
} mainarg_t;

extern mainarg_t  mainarg;

extern void getarg(int argc, char * argv[]);
```

Le programme 'pgmg' a maintenant deux arguments obligatoires et deux arguments optionnels:

```
pgmg [-c command_file_name] [-v] input_file_name output_file_name
```

Les deux arguments optionnels sont placés au début dans la ligne de commande mais dans n'importe quel ordre.

- -v est un drapeau qui demande au programme d'afficher un message à chaque étape du traitement.
- -c command_file_name permet de définir une série de traitement à appliquer à l'image, dans un fichier de commandes.

On se contente dans cette étape d'analyser les arguments de la ligne de commande. L'analyse des commandes contenues dans le fichier de commande sera réalisée à l'étape suivante.

La fonction getarg() doit traiter les cas d'erreurs. Une erreur possible est la détection d'un argument optionnel de type non défini (par exemple "-t"). Une autre erreur possible est la détection d'un nombre d'arguments supérieur à 4, ou inférieur à 2. En cas d'erreur, la fonction getarg() doit afficher un message d'erreur rappelant le format de la ligne de commande, et sortir du programme en utilisant l'appel système "exit".

Après avoir modifié le fichier "main.c" (en particulier pour introduire l'affichage des messages correspondant au mode 'verbose'), et après avoir écrit les fichiers "getarg.c" et "getarg.h", il vous reste à modifier le fichier "Makefile", pour prendre en compte ces nouveaux fichiers.

Compilez, puis exécutez le nouveau programme pgmg, en mode 'verbose', pour vérifier le bon fonctionnement.

Etape 3 : Exploitation du fichier de commandes

Commencez par créer un sous répertoire tme1/etape3, et recopiez vos fichiers source et votre Makefile dans ce répertoire, pour conserver les états intermédiaires de votre travail. Cette troisième étape consiste principalement à écrire un parseur élémentaire utilisant les fonctions fgets() et sscanf(). Vous pouvez vous inspirez de l'exemple que vous trouverez dans ScanfExample.

Le fichier de commandes contient une commande par ligne, et chaque commande déclenche une transformation complête de tous les pixels de l'image.

- S n : seuillage (si val > n, val = 255 / sinon val = 0)
- X : symétrie suivant x (x devient -x)
- Y : symétrie suivant y (y devient -y)
- P : rotation positive de 90°
- M : rotation négative de 90°
- F C00 C10 C20 C01 C11 C21 C02 C02 C12 C22 : filtrage produit de convolution avec une matrice 3*3

Voici un exemple de fichier de commandes:

```
# une ligne commencant par un # est un comentaire
X
Y
P
M
S 12
F 1 -1 1 2 10 2 1 -1 1
```

L'analyse du fichier de commande et l'exécution des transformations définies dans ce fichier sont réalisées par la fonction operate(). Cette fonction prend en entrée un pointeur sur le fichier de commande, et un pointeur sur une structure de donnée gmap contenant l'image initiale. Elle applique séquenciellement les transformations définies dans le fichier de commandes, et renvoie un pointeur sur la structure de donnée gmap contenant l'image résultante.

```
gmap *operate(FILE *commande, gmap *in);
```

Ecrivez le fichier operate.c, qui contient la fonction operate(), ainsi que le fichier associé operate.h. Cette fonction analyse le fichier de commande ligne par ligne, et exécute immédiatement la transformation demandée. Pour cela, on définira une fonction par type de commande. Ces 6 fonctions réalisant les transformations prennent pour nom le caractère définissant la commande, et possèdent au moins deux arguments: un pointeur sur l'image source, et un pointeur sur l'image résultat. Les deux structures de données doivent donc avoir été allouées en mémoire préalablement : pas de malloc() dans ces fonctions. Les 4 fonctions X(), Y(), P(), M() n'ont que 2 arguments, mais les deux fonctions S() et F() ont des arguments supplémentaires définissant les paramètres de la transformation:

```
void X(gmap *in, gmap *out);
void S(gmap *in, gmap *out, int n)
```

Pour cette étape, les fonctions de traitement n'exécutent pas réellement la transformation de l'image. Elles se contentent de recopier l'image in dans l'image out, et d'afficher un message:

```
Operation XXX non implementee
```

Après avoir écrit les fichiers "operate.c" et "operate.h", et après avoir modifié le fichier "main.c" pour prendre en compte le fichier de commandes, modifiez le fichier "Makefile", recompilez l'application et exécutez le programme pgmg en mode 'verbose' pour le fichier de commande fourni en exemple.

Etape 4 : Exécution des transformations S,X,Y,P,M,F

Implémentez successivement les 6 fonctions S(), X(), Y() P(), M(), F() dans le fichier "operate.c". Il faut calculer tous les pixels de l'image, en exécutant deux boucles for imbriquées pour parcourir les lignes et les colonnes. Pour alléger l'écriture, on définira la macro ELM(map,x,y) qui permet de désigner le pixel de coordonnées (x,y) dans l'image désignée par le pointeur map.

```
#define ELM(map,x,y)  map->raster[(y)*map->width + (x)]
```

Le coeur de ces fonctions sera toujours de la forme :

```
for(y = 0 ; y < height ; y++) {
  for(x = 0 ; x < width ; x++) {
    ELM(out, U(x,y), V(x,y)) = ELM(in, x, y);
  }
}
```

La difficulté est évidemment de définir les expressions arithmétiques U(x,y) et V(x,y) pour chacune des transformations à implémenter.

L'opération de filtrage peut s'exprimer plus simplement avec le dessin suivant:

Compte-rendu

La note de TME compte dans la note de contrôle continu. Vous n'avez pas de compte-rendu écrit à rendre, mais vous aurez à faire une démonstration de votre programme et à présenter le code au début de la séance de TME de la semaine suivante.

N'oubliez pas de commenter votre programme...