Mac2E

COLLABORATORS			
	TITLE : Mac2E		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 8, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Mac2E

1.1 Mac2E

```
*****************************************************************************

                                Mac2E (v3.0)
                    Préprocesseur de macros pour le langage E
                             Archive du 10 mars 1994
                    © Copyright 1993, 1994, Lionel Vintenat

*****************************************************************************

ATTENTION ! Tous les exécutables de cette archive nécessitent le Workbench 2.0
ou plus pour fonctionner. Désolé pour les utilisateurs du 1.3.
```


1.2 Introduction

```
    Ce paragraphe répond aux 3 questions essentielles :
    - Pourquoi Mac2E ?
    - Que fait Mac2E ?
    - Comment le fait-il ?
```

1.3 Comment tout a commencé...

Au début, il y avait l'Amiga E et moi. C'était formidable, à nous deux, on faisait de jolis programmes en un temps record. Comme je n'avais pas (et n'ai d'ailleurs toujours pas) les RKMs, ces programmes étaient trés laids, sans interface graphique, mais qu'importe, c'était le bon temps...

Et puis, MUI est arrivé, et là, plus rien n'a été pareil entre Amiga E et moi. Pourquoi ? Et bien Amiga E ne permet pas l'utilisation de macros, et programmer MUI sans macro, c'est presque de la folie ! D'autre part, c'était inconcevable de passer à côté de quelque chose comme MUI. Alors, je me suis rabattu un temps sur le langage C : ca a été le début des années noires pour mon Amiga...

Puis j'ai eu accés à INTERNET. J'ai donc parlé de mon problème à Wouter qui m'a conseillé d'utiliser un préprocesseur C : en voila une grande idée ! Mais après essais, ca s'est révélé trés lourd à utiliser : les temps de compilation étaient multipliés par 100, et surtout le compilateur n'affichait plus les bons numéros de lignes en cas d'erreur. C'est à ce moment là qu'est venu l'idée de Mac2E...

1.4 Présentation générale

Mac2E est un préprocesseur pour le compilateur Amiga E de Wouter van Oortmerssen, mais qui ne sait faire qu'une seule chose : remplacer des macros dans un source E. Autrement dit, les aspects "compilation conditionnelle" et "inclusion de fichiers" par exemple ne sont pas traités par Mac2E, alors que dans la plupart des préprocesseurs C, c'est le cas.

Ah ! J'allais oublié : tous les exécutables de cette archive sont bien-sûr écrits en Amiga E !

1.5 Esprit

J'ai conçu Mac2E avec 3 idées en tête :

- faire quelque chose de simple à utiliser (dans l'esprit de Amiga E)
- palier aux problèmes que j'avais rencontrés dans l'utilisation d'un préprocesseur C avec Amiga E (voir comment tout a commencé...)
- faire un préprocesseur dont l'utilisation ne rende pas les sources E dépendants de celui-ci; autrement dit, si une prochaine version d'Amiga E qui sort contient un préprocesseur, le passage de vos sources de Mac2E vers ce préprocesseur ne devra nécessiter que trés peu de modifications de ceux-ci

Et je pense que cette version 3.0 réalise effectivement ces 3 idées, à savoir :

- Mac2E reste trés proche au niveau de l'utilisation d'un préprocesseur C classique, donc son apprentissage sera trés rapide pour la plupart des programmeurs
- l'utilisation de Mac2E sur un fichier prend environ autant de temps que la compilation elle-même, ce qui, compte tenu de la rapidité de Amiga E lui-même, devrait être acceptable même pour les Amiga lents
- Mac2E n'introduit jamais de saut de ligne quand il remplace

une macro, ce qui fait que le compilateur indique toujours la bonne ligne en cas d'erreur
- la définition des macros se fait dans des fichiers à part des sources, et les fichiers de macros sont passés directement à la commande Mac2E sans que vos sources aient besoin d'être modifiés d'un caractère, donc le passage de Mac2E au futur (peut-être) préprocesseur de Amiga E sera trés simple et n'entrainera qu'un minimum de modifications de vos sources

1.6 Qu'est-ce qu'une macro ?

De manière grossière, on definit une macro en associant un identificateur (le nom de la macro) à une chaine de caractères (le corps de la macro). Ensuite, au lieu de mettre dans vos sources le corps de la macro, vous mettez simplement son nom, et c'est le préprocesseur qui remplacera le nom de la macro par son corps.

A mon avis, les macros sont très utiles dans 3 cas :
- pour eviter de réécrire plusieurs fois une même séquence
  -> voir exemple 1
- pour faciliter l'utilisation de valeurs abstraites
  -> voir exemple 2
- pour regrouper logiquement une sequence de programme
  -> voir exemple 3

Bien-sûr, ce qui précède n'est qu'une vue très superficielle de la notion de macro. Les macros telles qu'elles sont implantées aujourd'hui dans tous les préprocesseurs permettent beaucoup plus de choses. Les paragraphes suivants présentent en détail l'utilisation des macros.

1.7 Exemple 1

Prenons l'exemple d'un programme qui lit séquentiellement la mémoire. Pour cela, 2 variables sont définies :

```
DEF pointeur_memoire:PTR TO CHAR, caractere
```

L'accés à un octet se fera donc de la manière suivante :

```
caractere:=Char(pointeur_memoire++)
```

Sans macro, vous devrez donc écrire à chaque lecture la chaine précédente. Si vous faîtes des lectures dans plusieurs procédures, cela peut devenir rapidement fastidieux. La solution est de définir une macro de nom LireMemoire et de corps caractere:=Char(pointeur_memoire++). Il vous suffit alors d'écrire seulement LireMemoire à chaque fois.

1.8 Exemple 2

Pour ouvrir une librairie, il faut fournir à la fonction OpenLibrary() son nom obligatoirement en minuscules. En effet, si vous écrivez OpenLibrary('Dos.library',0), il n'y aura pas d'erreur à la compilation, mais la librairie ne sera pas trouvée à l'exécution. La solution est de définir une macro de nom NomDosLibrairie et de corps 'dos.library'. Comme cela, il vous suffit d'écrire OpenLibrary(NomDosLibrairie,0) pour ouvrir la dos.library, sans risque de vous tromper.

1.9 Exemple 3

Si vous voulez être sûr que stdout soit non nul, la documentation de Amiga E conseille d'écrire en début de programme WriteF(''). Une solution plus élégante est de définir une macro de nom OuvrirStdout et de corps WriteF(''). Après vous utilisez simplement OuvrirStdout dans vos sources,ce qui est beaucoup plus parlant.

Sur cet exemple très simple, la différence entre ce cas et les 2 précédents n'est pas très nette, mais ce qu'il faut bien voir, c'est que la macro OuvrirStdout n'est pas locale à un programme (contrairement à l'exemple 1), mais servira dans tous ceux où il sera nécessaire que stdout soit non nul. De plus, OuvrirStdout se comporte comme une mini-procédure (contrairement à l'exemple 2) qui accomplit une tâche.

1.10 Définir une macro

Les paragraphes qui suivent expliquent les différentes syntaxes de définition d'une macro, de la plus simple à la plus compliquée.

1.11 Définition d'une macro sans paramètre

Une définition simple de macro a la syntaxe suivante :

```
#define nom_macro corps_macro
   |   |   |     |     |     |
  (1) (2) (3)   (2)   (4)   (5)
```

où

(1) #define marque le début de la définition et peut se trouver n'importe où sur la ligne (pas forcément au début)

(2) 1 ou plusieurs espaces et tabulations

(3) le nom de la macro (combinaison quelconque de chiffres, de lettres majuscules et minuscules, et du caractère "_")

(4) le corps de la macro (combinaison quelconque de caractères différents du retour chariot)

(5) un retour chariot qui marque la fin de la définition de la macro

Exemples :

```
#define LireMemoire     caractere:=Char(pointeur_memoire++)
```

```
#define NomDosLibrairie 'dos.library'
#define OuvrirStdout    WriteF('')
```

1.12 Définition d'une macro avec paramètre(s)

Comme une procédure, une macro peut aussi avoir des paramètres. La syntaxe de définition est alors la suivante :

```
#define nom_macro(parametre1,parametre2,...,parametreN) corps_macro
   |   |   |      |    |      |    |      |   |    |      |       |     |
  (1) (2) (3)    (4)   |    (5)   |    (5) (5)   |    (7)     (8)    (9)
                       +----------+--------------+
                                  |
                                 (6)
```

où

(1) #define marque le début de la définition et peut se trouver n'importe où sur la ligne (pas forcément au début)

(2) 1 ou plusieurs espaces et tabulations

(3) le nom de la macro (combinaison quelconque de chiffres, de lettres majuscules et minuscules, et du caractère "_")

(4) parenthèse ouvrante directement accolée au nom de la macro

(5) virgule pour séparer chaque paramètre

(6) 1 ou plusieurs paramètres (combinaison quelconque de chiffres, de lettres majuscules et minuscules, et du caractère "_"), chaque paramètre pouvant être précédé et suivi d'un nombre quelconque d'espaces et de tabulations

(7) parenthèse fermante qui peut être suivi d'un nombre quelconque d'espaces et de tabulations

(8) le corps de la macro (combinaison quelconque de caractères différents du retour chariot)

(9) un retour chariot qui marque la fin de la définition de la macro

Exemples :

```
#define Puissance2( x )                 ((x)*(x))
#define EchangerVariablesXY(X,Y,TEMP)   TEMP:=X; X:=Y; Y:=TEMP
#define Max( x , y )                    (IF (x)>(y) THEN (x) ELSE (y))
```

Les paramètres précisés lors de la définition de la macro sont appelés paramètres formels.

1.13 Définition avancée d'une macro

Il se peut que le corps d'une macro soit trop long pour tenir sur une seule ligne de l'affichage. Il est alors possible de couper le corps en plusieurs morceaux. Pour indiquer au préprocesseur que le corps se continue à la ligne suivante, il faut placer un caractère "\" juste avant le retour chariot qui termine la ligne. A ce moment là, le préprocesseur sautera le caractère "\" et le retour chariot, et interprètera la ligne suivante dès son premier caractère, comme faisant partie du corps de la macro. Un corps de macro peut ainsi se poursuivre sur plusieurs lignes. La syntaxe de définition est dans ce cas la suivante :

```
#define nom_macro(parametres)   corps_morceau1 \
                corps_morceau2 \
```

```
                ...
                corps_morceauN
```

Attention, il faut obligatoirement que le caractère "\" soit immédiatement suivi par un retour chariot pour que le préprocesseur comprenne que le corps de la macro se poursuit à la ligne suivante.

1er exemple :

```
#define EchangerVariablesXY(X,Y,TEMP) TEMP:=X; \
X:=Y; \
Y:=TEMP
```

est une macro qui a pour corps TEMP:=X; X:=Y; Y:=TEMP
(notez que les caractères ";" étaient nécessaires car le préprocesseur a sauté les retours chariots après le caractère "\")

2ème exemple :

```
#define DireBonjour WriteF('Coucou, c\aest moi qui ai fait \
le super programme Mac2E (pub) !\n')
```

est une macro qui a pour corps

```
WriteF('Coucou, c\aest moi qui ai fait le super programme Mac2E (pub) !\n')
```

(notez que seul le caractère "\" suivi d'un retour chariot a été interprété comme un signe de continuation du corps)

3ème exemple :

```
#define MacroInutile [1 espace ->\
 ][2 espaces ->\
  ][3 espaces ->\
   ] et c'est tout !
```

est une macro qui a pour corps

```
[1 espace -> ][2 espaces ->  ][3 espaces ->   ] et c'est tout !
```

1.14 Utiliser une macro

Le tout n'est pas de définir des macros, encore faut-il pouvoir les utiliser ! Pour cela, il suffit de placer les noms des macros que vous avez définies où vous en avez besoin dans le fichier source, comme vous le feriez pour des instructions normales. Mais attention, une macro n'est pas une instruction reconnue par le compilateur. Avant de compiler un fichier qui contient des macros, vous devez utiliser un préprocesseur. Le rôle de ce programme est de trouver tous les noms de macro d'un fichier source, et de remplacer ces noms par le corps de la macro associée. Le corps d'une macro doit lui contenir des instructions reconnues par le compilateur ! Une fois le travail du préprocesseur achevé, le fichier est compilable.

Les paragraphes suivants expliquent en détail comment le préprocesseur procède pour trouver et remplacer un nom de macro.

- Identifier un nom de macro
- Traiter le passage d'arguments
- Remplacer une macro

1.15 Identifier un nom de macro

Pour qu'un nom de macro soit reconnu par le préprocesseur, il faut que le nom que vous avez écrit dans le fichier source soit :
- exactement le même que celui précisé lors de la déclaration, en particulier, le préprocesseur fait la différence entre majuscules et minuscules
- précédé et suivi d'un caractère différent d'une lettre, d'un chiffre et du caractère "_"

Si les 2 conditions précédentes sont remplies, le préprocesseur reconnaitra le nom de la macro.

Exemples :
Supposons que vous ayez défini une macro de nom toto (peu importe son corps). Elle sera reconnue dans les séquences d'instructions suivantes :

```
a:=toto+1
WriteF('Chaine bidon pour introduire \d !\n',toto)
```

Par contre, le préprocesseur ne la reconnaitra pas dans les séquences d'instructions suivantes :

```
a:=different_de_toto+1
WriteF('Chaine bidon pour introduire \d !\n',toto1)
```

1.16 Traiter le passages d'arguments

Vous avez vu dans une section précédente que l'on pouvait donner à une macro des paramètres (dits formels) lors de sa définition, comme vous le feriez pour une procédure. Et bien comme pour une procédure, quand vous utilisez une macro ainsi définie dans un fichier source, vous devez lui fournir des arguments (dits paramètres réels). La syntaxe d'appel d'une macro (j'utilise le mot appel par analogie avec les procédures) est la suivante :

```
nom_macro(parametre1,parametre2,...,parametreN)
   |     |    |     |    |     |   |    |     |
  (1)   (2)   |    (4)   |    (4) (4)   |    (5)
              +----------+--------------+
                         |
                        (3)
```

où
(1) le nom de la macro
(2) parenthèse ouvrante directement accolée au nom de la macro
(3) 1 ou plusieurs paramètres (combinaison quelconque de caractères différents du retour chariot)
(4) virgule pour séparer chaque paramètre
(5) parenthèse fermante

Attention, les paramètres sont délimités par des virgules et des parenthèses, et entre 2 de ces symboles consécutifs, tous les caractères sont pris en compte, et interprétés comme faisant partie d'un paramètre.

Si une macro a été définie sans paramètres, sa syntaxe d'appel est tout simplement nom_macro.

Quand le préprocesseur analyse un appel de macro, il s'attend bien-sûr à trouver pour cette macro autant de paramètres réels que de paramètres formels ! En particulier, une macro définie sans arguments ne doit pas être suivi par un caractère "(", sinon le préprocesseur croiera que cette macro est appelée avec des arguments.

Si la syntaxe d'appel d'une macro est correcte, le préprocesseur associe alors à chaque paramètre formel le paramètre réel correspondant, comme le compilateur le fait pour une procédure.

Exemples :
Supposons que vous ayez défini une macro toto ainsi :

```
#define toto(param1, param2) corps_quelconque
```

Voici un tableau de ce qui se passera pour plusieurs séquences d'appel :

séquences d'appel	associé à param1	associé à param2
toto(a,1)	a	1
toto(a , 1)	a	1
toto((3+2)*5 ,WriteF('Ah !\n'))	(3+2)*5	WriteF('Ah !\n')
toto (a,1)	E R R E U R	
toto(1,2,3)	E R R E U R	

1.17 Remplacer une macro

Si la macro à remplacer a été définie sans paramètre, alors le préprocesseur substitue simplement le nom de cette macro par son corps.

Si par contre, la macro à remplacer a été définie avec des paramètres, le préprocesseur remplace aussi le nom de la macro par son corps, mais en substituant tous les paramètres formels du dit corps par les paramètres réels correspondants.

1er exemple :
Considérons la définition de macro suivante :

```
#define NomDosLibrairie 'dos.library'
```

Alors on aura par exemple l'appel OpenLibrary(NomDosLibrairie) qui sera remplacé par OpenLibrary('dos.library').

2ème exemple :
Considérons les définitions de macro suivantes :

```
#define Square(x) ((x)*(x))
#define Max(x,y) (IF (x)>(y) THEN (x) ELSE (y))
```

Alors on aura par exemple l'appel a:=Square(4+3) * Max(7,2*(8-2)) qui sera remplacé par a:=((4+3)*(4+3)) * (IF (7)>(2*(8-2)) THEN (7) ELSE (2*(8-2)))

Notez comment les nombreuses parenthèses présentes dans les corps des 2 macros précedentes forcent la priorité d'évaluation de cette expression. Sans elles, le résultat obtenu ne serait surement pas celui attendu. D'une manière générale, il faut faire très attention lors de la création d'une macro. En effet, même si une macro ressemble à une procédure ou à une fonction, ce n'est pas la même chose ! Le corps d'une macro n'est jamais évalué lors d'un appel, il est simplement substitué au nom de la macro. Il peut donc se retrouver directement accolé à une autre expression. L'exemple de la macro Square met bien en évidence ce genre de problème.

1.18 Usage avancé

Arrivé à cette section, vous devriez maintenant bien maîtriser la définition et l'usage des macros. Si ce n'est pas le cas, revenez en arrière.
Les paragraphes suivants regroupent donc des aspects plus techniques de l'utilisation des macros, mais qu'il est quand-même important de connaître.

1.19 Macros, commentaires et chaines de caractères

Il a été dit précédement qu'un appel de macro pouvait se trouver n'importe où dans un fichier source. Et bien ce n'est pas vrai ! En effet, le préprocesseur ne recherche pas les appels de macro dans les commentaires (même imbriqués) et les chaines de caractères. En effet, les macros sont là pour regrouper sous un même nom un bout de code de programme. Il n'y a donc aucune raison de mettre des appels de macro à l'intérieur.
En pratique, cela signifie que vous pouvez mettre ce qu'il vous plait en commentaire et dans les chaines de caractères, le préprocesseur n'y touchera pas.

1.20 Des macros dans un corps de macro

Quand vous définissez une macro, vous pouvez mettre ce que voulez dans le corps de celle-ci, même des appels à d'autres macros. Le préprocesseur traitera aussi ce genre d'appels internes à un corps, lors de la substituion du nom de la macro englobante. En fait, le préprocesseur fait toute les substitutions possibles, et après son passage, il ne reste plus un seul appel de macro dans le fichier source. Bien-sûr, les arguments d'un appel interne à un corps de macro peuvent être des paramètres formels de la macro elle-même. Il n'y a pas de limite à la profondeur de ces imbrications.
Attention, un corps de macro ne peut contenir d'appel à elle-même, sinon le préprocesseur fera infiniment le même remplacement, jusqu'à épuisement de la mémoire...ou de la patience de l'utilisateur !

1er exemple :
Supposons que vous définissiez 2 macros ainsi :

```
#define ValeurInfinie $FFFFFFFF
#define NombreFiniPositif(x) (((x)>0) AND ((x)<>ValeurInfinie))
```

Alors, on aura par exemple l'appel
`IF NombreFiniPositif(A*B)=FALSE THEN WriteF('Erreur !\n')` qui sera remplacé par
`IF (((A*B)>0) AND ((A*B)<>$FFFFFFFF))=FALSE THEN WriteF('Erreur !\n').`

2ème exemple :
Supposons que vous définissiez 2 macros ainsi :

```
#define ValeurAbsolue(x) (IF (x)>0 THEN (x) ELSE -(x))
#define MaxDesValeursAbsolues(x,y) (IF ValeurAbsolue(x)>ValeurAbsolue(y) THEN
(x) ELSE (y))
```

Alors, on aura par exemple l'appel `a:=MaxDesValeursAbsolues(5,-(A*B))`

```
qui sera remplacé par
(IF (IF (5)>0 THEN (5) ELSE -(5))>(IF (-(A*B))>0 THEN (-(A*B)) ELSE -(-(A*B)))
THEN (5) ELSE (-(A*B))).
```

1.21 Appels de macro en argument d'une macro

Dans les exemples précédents, vous avez sans doute remarqué que les arguments passés à une macro pouvaient être quelconques, tout en restant cohérents bien-sûr (le retour chariot n'est par exemple pas permis dans un argument). Vous pouvez donc tout à fait mettre un appel de macro dans un argument d'une autre macro. Là encore, le préprocesseur traitera d'abord l'appel de macro inclus dans l'argument, puis ensuite l'appel de macro englobant avec l'argument traité. Il n'y a pas de limite à la profondeur de ce genre d'imbrications.

Rappelez-vous : la règle générale est que le préprocesseur traite absolument tous les appels de macro dans un fichier source, et cela où qu'il se trouvent, excepté dans les commentaires et les chaines de caractères.

1er exemple :

Considérons les définitions de macro suivantes :

```
#define ValeurBidon 12
#define Doubler(x) (2*(x))
```

Alors l'appel de macro suivant Doubler(ValeurBidon) sera remplacé par (2*(12)).

2ème exemple :

Considérons les définitions de macro suivantes :

```
#define MasquerPoidsFort(x) ((x) AND $FFFF)
#define Moyenne(x,y) (((x)+(y))/2)
```

Alors l'appel de macro suivant Moyenne(100,MasquerPoidsFort(100000)) sera remplacé par (((100)+((100000) AND $FFFF))/2).

1.22 Caractères spéciaux

Vous vous êtes sans doute demandé ce qui se passerait si un argument d'une macro contenait les caractères "(", ")" ou ",". En effet, comme ces derniers sont utilisés pour délimiter les arguments, leur présence peut prêter à confusion pour leur reconnaissance. Et bien, il n'en est rien : le préprocesseur est suffisament intelligent pour distinguer lesquels de ces caractères sont effectivement là pour délimiter des arguments, et lesquels appartiennent à ces mêmes arguments.

Attention, les arguments doivent quand-même rester cohérents ! Par exemple, à toute parenthèse ouvrante doit correspondre une parenthèse fermante. De même, une virgule doit se trouver entre guillemets ou dans une chaine de caractères entourée de quotes.

Exemples :

Considérons une macro toto définie comme ayant 2 paramètres formels. Voici un tableau de ce qui se passera pour différentes séquences d'appel :

séquences d'appel	1er paramètre	2ème paramètre

```
| toto((3+4)*(5-6),'1, 2 et 3')             | (3+4)*(5-6)    | '1, 2 et 3'    |
| toto((((())())()),caractere ",")          | (((())())())   | caractere ","  |
| toto(),4)                                 |          E R R E U R          |
| toto(4,,)                                 |          E R R E U R          |
+-------------------------------------------+----------------+----------------+
```

1.23 Utilisation de Mac2E

Pour comprendre ce qui va suivre, vous devez bien connaître ce qu'est une macro, et plus particulièrement comment définir et utiliser une macro comme cela se fait en langage C. Si ce n'est pas le cas, retournez à la section qu'est-ce qu'une macro ? . Si vous connaissiez déjà tout sur les macros du langage C avant d'avoir ce programme, la lecture de cette section n'est pas nécessaire. Néanmoins, vous devrez vous y reporter pour vérifier un point de syntaxe. En effet, les paragraphes qui suivent traitent de l'utilisation de PreMac2E et de Mac2E seule, sans rien rappeler sur les macros.

Le but de ces programmes est, je le rappelle, de permettre l'utilisation de macros dans vos sources.

La première chose à faire est donc de définir des macros. Cela se fait dans un fichier à part de vos sources, appelé fichier de macros. Ensuite, vous devez préanalyser ce fichier avec PreMac2E. Enfin, vous pouvez ensuite utiliser Mac2E pour remplacer dans vos fichiers sources les appels de macro. Le traitement d'un fichier source nécessite donc 3 étapes, que décrivent en détail les paragraphes suivants.

1.24 Les fichiers de macros

Un fichier de macro est un fichier ASCII ne contenant que des définitions de macros. Je rappelle que vous ne pouvez pas définir de macro dans vos fichiers sources, vous devez le faire à part, dans un fichier de macros.

Ces fichiers peuvent aussi contenir des commentaires. Ceux-ci peuvent être placés n'importe où, sauf à l'intérieur d'une définition de macro. Autrement dit, les commentaires se trouvent entre les définitions de macro. Notez que les commentaires peuvent être placés dans le fichier tels quels, sans marque de début ni de fin.

Normalement, les fichiers de macros sont placés dans le sous-répertoire MacroFiles/ de l'endroit où vous avez installé Amiga E.

Voir définir une macro

1.25 Appel de PreMac2E

Quand vous avez défini un fichier de macros, celui-ci n'est pas directement utilisable par Mac2E, le préprocesseur de cette archive. En effet, vous devez d'abord le pré-analyser. Ceci est réalisé par PreMac2E dont la syntaxe d'appel est la suivante :
PreMac2E fichier_de_macros fichier_de_macros_pre_analysé
où
- fichier_de_macros désigne un fichier de macros (éventuellement avec son chemin d'accés)
- fichier_de_macros_pre_analysé désigne le fichier (éventuellement avec son chemin d'accés) résultat de la pré-analyse

Par exemple, pour pré-analyser le fichier de macros mui.e fourni avec cette archive, j'ai tapé PreMac2E MacroFiles/mui.e PreAnalysedMacroFiles/mui.e.

Normalement, les fichiers de macros pré-analysés sont placés dans le sous-répertoire PreAnalysedMacroFiles/ de l'endroit où vous avez installé Amiga E.

Pendant une pré-analyse, PreMac2E cherche à remplacer tous les appels de macro internes à des corps d'autres macros. De même, il trie les macros puis les classe dans une table dite hash-codée. Enfin, il écrit le contenu de cette table dans le fichier résultat. Ainsi, quand Mac2E traitera un fichier source, il se servira, non pas du fichier de macros original, mais du fichier de macros pré-analysé. Le gain de vitesse sera ainsi énorme.

La pré-analyse d'un fichier de macro ne se fait donc qu'une seule fois (pourvu qu'il ne reste pas d'erreur dans ce fichier bien-sûr). Après, vous n'utilisez plus que le fichier pré-analysé. Si vous modifiez le fichier de macros original, vous devez évidemment refaire la pré-analyse avec PreMac2E pour que les changements soient pris en compte.

Notez que le format d'entrée exacte de PreMac2E est "FROM/A,TO/A,VER=VERBOSE/S,KS=KEEPSPACES/S". Vous pouvez donc préciser 2 paramètres supplémentaires lors de son appel.

VERBOSE est décrit dans la section sur les messages d'erreur Messages_erreur}.

KEEPSPACES force PreMac2E à conserver dans le corps des macros les espaces et les tabulations situés en début de ligne, lorsqu'un corps se poursuit sur plusieurs lignes. Par défaut, PreMac2E les élimine, ce qui diminue la taille du fichier pré-analysé, et accélère le traitement de Mac2E.

1.26 Appel de Mac2E

Mac2E est le véritable préprocesseur de cette archive. C'est en effet lui qui s'occupe de remplacer dans vos fichiers sources les appels de macro par le corps aproprié. Sa syntaxe d'appel est la suivante :
Mac2E fichier_source_e fichier_destination_e liste_fichiers_pré_analysés
où
- fichier_source_e est un fichier source (avec éventuellement son chemin d'accés) qui contient des appels de macro à traiter
- fichier_destination_e est un fichier (avec éventuellement son chemin d'accés) qui contiendra le fichier_source_e mais avec les appels de macro traités
- liste_fichiers_pré_analysés est une liste de 1 à plusieurs fichiers de macros pré-analysés (avec éventuellement leur chemin d'accés)

Le format d'entrée exacte de Mac2E est "FROM/A,TO/A,WITH/A/M".

Appelé ainsi, Mac2E charge tous les fichiers de macros pré-analysés

précisés dans la ligne de commande, puis à partir des macros qui y sont définies, traite le fichier source.

Voir utiliser une macro et usage avancé

1.27 Les messages d'erreur

Tous les messages d'erreur retournés par PreMac2E et Mac2E sont suffisament explicites par eux-mêmes. Le numéro de ligne où se trouve l'erreur est également precisé.

La seule exception à cela est quand PreMac2E traite dans les corps des macros qu'il a trouvées dans un fichier de macros, les appels internes à d'autres macros. En effet, cette phase de la pré-analyse se fait une fois toutes les définitions de macro enregistrées. Donc PreMac2E ne travaille plus à ce moment là sur des numéros de ligne. Pour connaître l'endroit où se situe une erreur signalée pendant cette phase, il faut relancer PreMac2E avec l'option VERBOSE. Cela va forcer PreMac2E à afficher tous les noms de macro dont il traite le corps. Ainsi, quand il affiche une erreur, il vous suffit de regarder le nom de la macro dont il traitait le corps quand l'erreur est survenue, et de trouver cette macro dans le fichier. L'erreur est dans le corps de celle-ci.

1.28 Mac2E et MUI

Si vous avez lu comment tout a commencé... , vous savez que Mac2E doit son existence à ce que MUI est infiniment plus facile à programmer avec des macros que sans. C'est pourquoi le 1er exemple (et le seul pour l'instant) d'utilisation de Mac2E va concerner MUI. Vous trouverez dans cette archive tout ce qu'il faut pour utiliser MUI avec Amiga E, pratiquement de la même manière que vous le feriez en C. Pour cela, il faut 6 choses :

- PreMac2E
- Mac2E
- mui.m
- muimaster.m
- mui.e
- OptiMUI2E

L'idée générale qui a gouverné la conception de mui.m et mui.e est je le rappelle de faire "une interface MUI-Amiga E" très proche de celle du C, donc pendant la programmation, il est utile de se baser sur le fichier include mui.h destiné au langage C. En effet, celui-ci contient un nombre impressionnant de commentaires, qui ne sont pas tous présents dans "la version E".

Toute "cette interface" est basée sur MUI 2.0. Dans l'archive de MUI, il y a déjà certains fichiers pour l'utiliser en langage E, mais en aucun cas, aussi complets ni aussi pratiques que ceux fournis ici. Oubliez les donc et essayez ceux-là !

```
Mac2E
PreMac2E
mui.m
muimaster.m
```

mui.e
OptiMUI2E

1.29 mui.m

mui.m est comme son nom l'indique un fichier include classique de Amiga E. Il contient toutes les structures définies dans mui.h à la différence près que tous les noms (des structures et des champs de celles-ci) sont en minuscules. Cette limitation est due à Iconvert.

Donc pour utiliser dans vos programmes des structures MUI, il vous faut mettre au début de votre fichier source MODULE 'libraries/mui.m'.

1.30 muimaster.m

muimaster.m est comme son nom l'indique un fichier include classique de Amiga E. Il contient toutes les définitions des fonctions de la librairie muimaster.library. Les noms des fonctions sont les mêmes qu'en C excepté qu'ils commencent tous par Mui au lieu de MUI (exemple : Mui_NewObjectA). Cette limitation est imposée par Amiga E car les noms de fonction doivent avoir leur première lettre en majuscule et la deuxième en minuscule.

Donc pour utiliser dans vos programmes des fonctions de la librairie muimaster.library (et il y a des chances que ce soit le cas !), il vous faut mettre au début de votre fichier source MODULE 'muimaster.m'.

1.31 mui.e

mui.e est la clé de voute de "cette interface MUI-Amiga E" puisqu'il contient toutes les macros (constantes et choses plus complexes comme la définition d'objets) du fichier mui.h, mais adaptées pour le langage E. La syntaxe des macros de mui.e, ainsi que la syntaxe de leur corps, est exactement la même que dans mui.h.
En plus de l'intérêt pour l'utilisation de MUI de ce fichier, il constitue aussi une grosse bibliothèque d'exemples de définition de macros.

Le fichier mui.e est fourni en 2 exemplaires : un dans le répertoire MacroFiles/ et l'autre dans le répertoire PreAnalysedMacroFiles/. Le premier d'entre eux est donc une version textuelle lisible, au contraire du second qui a été pré-analysé avec PreMac2E.
Donc pour utiliser toutes ces macros MUI dans vos sources E, il faut lancer Mac2E sur votre fichier source avant l'appel du compilateur :
Mac2E source.e destination.e PreAnalysedMacroFiles/mui.e

1.32 OptiMUI2E

Si vous jetez un coup d'oeil sur mui.e, vous verrez dans le corps des macros qui définissent de nouveaux objets des "TAG_IGNORE, 0", par exemple "#define WindowObject MuI_NewObjectA('Window.mui', [TAG_IGNORE, 0". Ce tag ne fait comme son nom l'indique strictement rien à l'exécution. Cependant, j'ai été obligé de les introduire pour garder la même syntaxe d'utilisation qu'en C. C'est à ce niveau qu'intervient OptiMUI2E. Son rôle est d'enlever ces "TAG_IGNORE, 0" inutiles des sources E. Sa syntaxe d'appel est la suivante :
OptiMUI2E fichier_source_e fichier_destination_e
où
- fichier_source_e désigne le nom d'un fichier source (avec éventuellement son chemin d'accés) où il y a des "TAG_IGNORE, 0" à enlever
-fichier_destination_e désigne le nom d'un fichier (avec éventuellement son chemin d'accés) qui contiendra fichier_source_e avec tous les "TAG_IGNORE, 0" supprimés
Plus généralement, le format d'entrée de OptiMUI2E est "FROM/A,TO/A".

Attention, OptiMUI2E enlève parfois des retours chariots de vos sources pour respecter la coupure des lignes sur une virgule, obligatoire en E. Donc le fichier produit ne comporte pas forcément le même nombre de lignes que le fichier de départ, d'où de possibles problèmes pour les numéros de ligne d'erreur retournés par le compilateur. Il est donc trés vivement conseillé de n'utiliser OptiMUI2E que pour une ultime compilation une fois le programme terminé et testé. De toute façon, OptiMUI2E n'est absolument pas nécessaire pour utiliser MUI avec Amiga E. Il réduit un peu la taille des sources et des exécutables utilisant des macros MUI, mais ceci dans une faible mesure.

1.33 Bugs

Pas de panique, tous les points suivants ne sont pas des bugs, mais plutôt des limitations.

* PreMac2E ne vérifie pas si les déclarations de macros sont récursives, donc si c'est le cas, PreMac2E bouclera...
* PreMac2E et Mac2E ne verifient pas si une macro est definie plus d'une fois. Si c'est le cas, Mac2E en utilisera une, mais laquelle, voilà la question ?!?!
* La longueur du nom d'une macro est limitée à 255 caractères.
* Le nombre d'arguments est limité à 32.
* La longueur du corps d'une macro avant pré-analyse est limitée à 4Ko. Si cela ne vous suffit pas, ce n'est pas d'une macro dont vous avez besoin, c'est d'une procédure !
* La longueur du corps d'une macro après pré-analyse est limitée à 64Ko, et là, ca devrait vous suffire, non ?
* PreMac2E ne vérifie pas que les 4 limitations précédentes sont respectées.

1.34 Historique

```
Version 1.0 :  - 1ère version fonctionnelle de Mac2E (TRES TRES LENTE...)
Version 2.0 :  - version remaniée de la v1.0 avec beaucoup d'optimisations
                   assembleur dans le source E (10 fois plus rapide !)
```

```
               - adjonction de OptiMUI2E v1.0
               - 1ère version distribuée
Version 3.0 :  - adjonction de PreMac2E v1.0 pour pré-analyser les fichiers
                   de macros
               - utilisation d'une table hash-codée (14 fois plus rapide !)
               - PreMac2E et Mac2E renvoie maintenant des messages d'erreur
                   explicites
               - vérification de toutes les allocations mémoires
               - quelques bugs mineurs corrigés
               - OptiMUI2E v1.1 marche en 68000
               - mui.e est maintenant commenté
               - source de la fonction doMethod() fourni
               - les sources de tous les exécutables sont fournis
               - une meilleure documentation
               - mise à jour de mui.e par rapport à MUI v2.0
Version 3.1 :  - quelques bugs mineurs corrigés
               - mise à jour de mui.e par rapport à MUI v2.1
```

1.35 Futur

J'attends (comme vous) la prochaine version d'Amiga E qui ne devrait plus tarder d'après Wouter... Bien-sûr, j'attends également vos suggestions !

1.36 Distribution

Cette archive peut être librement distribuée, tant que personne ne tire aucun bénéfice de cette distribution. Tout autre type de vente ne peut en aucun cas être effectué sans l'autorisation de l'auteur.

Cette archive peut être incluse dans des collections de logiciels du domaine public, tant que les conditions ci-dessus restent vérifiées.

Cependant, l'archive doit être distribuée dans son entier et doit avoir la structure suivante :

```
Mac2E/Bin/Mac2E
Mac2E/Bin/OptiMUI2E
Mac2E/Bin/PreMac2E
Mac2E/Docs/Mac2E.guideD
Mac2E/Docs/Mac2E.guideD.info
Mac2E/Docs/Mac2E.guideE
Mac2E/Docs/Mac2E.guideE.info
Mac2E/Docs/Mac2E.guideF
Mac2E/Docs/Mac2E.guideF.info
Mac2E/MacroFiles/mui.e
Mac2E/MacroFiles/mui.e.info
Mac2E/Modules/libraries/mui.m
Mac2E/Modules/muimaster.m
Mac2E/PreAnalyzedMacroFiles/mui.e
Mac2E/Sources/doMethod.e
Mac2E/Sources/doMethod.e.info
Mac2E/Sources/Mac2E.e
Mac2E/Sources/Mac2E.e.info
Mac2E/Sources/OptiMUI2E.e
Mac2E/Sources/OptiMUI2E.e.info
```

```
    Mac2E/Sources/PreMac2E.e
    Mac2E/Sources/PreMac2E.e.info
    Mac2E/Liesmich.zuerst
    Mac2E/Liesmich.zuerst.info
    Mac2E/LisezMoi.d_abord
    Mac2E/LisezMoi.d_abord.info
    Mac2E/ReadMe.first
    Mac2E/ReadMe.first.info
    Mac2E/ReadMe.mui
    Mac2E.info
```

D'autre part, tous ces fichiers (exceptés mui.m, muimaster.m, mui.e, ReadMe.mui et toutes les icones), restent sous copyright de l'auteur. Aucun d'entre eux ne peut être modifié sans mon accord.

Enfin je dégage toute responsabilité quant à l'utilisation de ce programme et les dommages qu'il pourrait causer : vous l'utilisez à vos risques et périls !

Ce programme est distribué selon le concept Freeware, c'est-à-dire que vous n'étes absolument pas obligé de m'envoyer quoi que ce soit ! Cependant je serais heureux de recevoir n'importe quoi, d'un Amiga 4000/40 à une simple carte postale de vos vacances, en passant par 20FF ou un simple e-mail ! (voir l'auteur)

1.37 L'auteur

Vous pouvez me joindre par courrier :

- à mon adresse étudiante valable jusqu'en juillet 1994 inclus :

```
Lionel Vintenat
appartement 21
11 rue François Oulié
31500 TOULOUSE
FRANCE
```

- à mon adresse familiale :

```
Lionel Vintenat
3 impasse Boileau
Lotissement Les Termes
87270 COUZEIX
FRANCE
```

Ecrivez moi plutôt à mon adresse étudiante jusqu'en juillet 1994 car j'y suis beaucoup plus souvent qu'à mon domicile familiale.

Vous pouvez également me joindre sur INTERNET. Mon adresse e-mail est vintenat@irit.fr. Je préfère de trés loin que vous me contactiez par e-mail que par courrier. Je répondrai toujours aux questions qui me seront posées par e-mail, par contre n'espérez pas de réponses à un courrier (je suis trés fainéant dés qu'il s'agit de prendre un stylo...).

1.38 Les remerciements

Un grand merci :
- à l'Amiga pour être le meilleur ordinateur personnel
- à Wouter van Oortmerssen pour son travail dans le domaine de la compilation (essayez son FALSE, surprise garantie !) en général et pour Amiga E en particulier
- à Brian Mury pour la traduction de la doc en anglais
- à Marc Schröer pour la traduction de la doc en allemand
- à Xavier Billault pour son aide dans la conception de cette documentation
- à tous ceux de la mailing liste Amiga française qui m'ont aidé
- à tous ceux qui font du domaine public en général

Enfin, merci d'avance à tous ceux qui me signaleront des bugs ou des suggestions, ou encore qui me feront parvenir des corrections ou des traductions de cette documentation (voir l'auteur).

Bonne programmation en E and ...

NEVER FORGET, ONLY AMIGA MAKES IT POSSIBLE !