Mac2E

COLLABORATORS			
	TITLE : Mac2E		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		November 28, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Mac2E

1.1 Mac2E

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

                                Mac2E (v4.0)
                    Préprocesseur de macros pour le langage E
                           Archive du 2 septembre 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 très 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 très 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 rencontrès 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 très peu de modifications de ceux-ci

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

- Mac2E reste très proche au niveau de l'utilisation d'un préprocesseur C classique, donc son apprentissage sera très 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 très 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

Plus fort encore pour définir une macro...

Utiliser plusieurs lignes pour le corps
Des commentaires dans le corps

1.14 Utiliser plusieurs lignes pour le corps

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 !
```

1.15 Des commentaires dans le corps

A partir de Mac2E v4.0, vous pouvez placer des commentaires à l'intérieur du corps d'une macro. Ceux-ci sont introduits par les deux caractères "->" et se terminent avec le retour chariot en fin de ligne (ou la fin du fichier éventuellement). Le principe est donc le même que pour les nouveaux commentaires d'Amiga E v3.0 (l'idée vient d'ailleurs de là !). Tout commentaire est bien sûr ignoré par le préprocesseur.

1er exemple :

```
#define NomMacro CorpsMacro -> Ceci est un commentaire
```

est une macro qui a pour corps CorpsMacro .

De la même manière que pour une macro dont le corps ne comporte pas de commentaire (voir Utiliser plusieurs lignes pour le corps), vous pouvez étendre le corps d'une macro dont le corps comporte des commentaires sur plusieurs lignes.

2ème exemple :

```
#define NomMacro PremierMorceauDuCorps  -> Premier commentaire  \
                 DeuxièmeMorceauDuCorps -> Deuxième commentaire \
                 DernierMorceauDuCorps  -> Dernier commentaire
```

est une macro qui a pour corps

```
PremierMorceauDuCorps  DeuxièmeMorceauDuCorps DernierMorceauDuCorps  .
```

1.16 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

1.17 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)
```

1.18 Traiter le passage 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, et qui peut être suivi d'un nombre quelconque de retours chariots (à partir de Mac2E v4.0 seulement)

(3) 1 ou plusieurs paramètres (combinaison quelconque de caractères différents du retour chariot)

(4) virgule pour séparer chaque paramètre, et qui peut être suivi d'un nombre quelconque de retours chariots (à partir de Mac2E v4.0 seulement)

(5) parenthèse fermante, qui peut être précédé par un nombre quelconque de retours chariots (à partir de Mac2E v4.0 seulement)

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, à l'exception des retours chariots qui suivent la parenthèse ouvrante, suivent les virgules et précèdent la parenthèse fermante (à partir de Mac2E v4.0 seulement).

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           |
+--------------------------------------+-----------------+-----------------+
```

1.19 Remplacer une macro

Si la macro à remplacer a été définie sans paramètre, alors le prépro-
cesseur 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.

1.20 Usage avancé

Arrivé à cette section, vous devriez maintenant bien maîtriser la défi-
nition 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.

1.21 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.

1.22 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))).
```

1.23 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'imbri-

cations.
    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).

1.24 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.

    Depuis Mac2E v4.0, les crochets "[" et "]" sont traités de la même manière que les parenthèses. Vous pouvez donc passer une liste E en paramètre d'une macro. Les règles de cohérence s'appliquent aussi pour les crochets : pour chaque crochet ouvrant, il doit exister dans le paramètre un crochet fermant.

    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([1,(),'str'],caractere ",")	[1,(),'str']	caractere ","
toto(),4)	E R R E U R	
toto(4,,)	E R R E U R	

1.25 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 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 des fichiers à part de vos sources, appelés fichiers de macros.
Ensuite vous pouvez utiliser Mac2E pour remplacer dans vos fichiers sources les appels de macro.

1.26 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 à l'extérieur 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.
A partir de Mac2E v4.0, on peut aussi placer des commentaires à l'intérieur de la définition d'une macro, mais ceux-ci obéissent à une syntaxe particulière (voir des commentaires dans le corps).

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

1.27 Appel de Mac2E

La syntaxe d'appel de Mac2E est la suivante :
FROM/A,TO/A,WITH/M,PA=PREANALYZE/S,VER=VERBOSE/S,KS=KEEPSPACES/S,DEBUG/S.

En fait, le comportement précis de Mac2E est déterminé par le drapeau PREANALYZE (PA en abrégé). Si celui-ci est positionné, Mac2E va pré-analyser un fichier de macros, dans le cas contraire, il va se comporter comme un préprocesseur.

@{ " Mode pré-analyse " Link Mode_préanalyse }
@{ " Mode préprocesseur " Link Mode_préprocesseur }

```
@{ " VERBOSE            " Link VERBOSE              }
@{ " KEEPSPACES         " Link KEEPSPACES           }
@{ " DEBUG              " Link DEBUG                }
```

Il est possible d'interrompre Mac2E à tout moment et quel que soit son mode de fonctionnement en tapant Ctrl-C.

1.28 Mode pré-analyse

Dans ce mode, Mac2E va pré-analyser le fichier de macros précisé dans le champ FROM et sauver le résultat de la pré-analyse dans le fichier précisé dans le champ TO. Le champ WITH est donc inutile et doit donc être vide lors de l'appel de Mac2E.

L'intérêt de pré-analyser un fichier de macros est bien sûr le gain de vitesse. En effet, pendant une pré-analyse, Mac2E fait principalement 3 choses :

- traiter dans les corps de toutes les macros les appels d'autres macros
- classer les macros dans une table hash-codée
- repérer précisement la position de tous les arguments dans le corps des macros

Ainsi, quand Mac2E fonctionnera en mode préprocesseur, si il utilise un fichier de macros pré-analysé au lieu du fichier de macros original, il disposera du corps de toutes les macros prêts à être insérés dans le fichier source sans aucune analyse supplémentaire à faire.

Cette technique présente cependant deux désavantages :

- après chaque modification d'un fichier de macros, il faut mettre à jour le fichier de macros pré-analysé correspondant avec une nouvelle pré-analyse
- un fichier de macros pré-analysé est utilisé par Mac2E en mode préprocesseur tel quel sans aucune analyse supplémentaire : ainsi si vous appelez Mac2E en mode préprocesseur avec deux fichiers de macros, dont un est pré-analysé, Mac2E ne vérifiera pas que ce dernier contient des macros déjà définies dans l'autre ou que des macros de ce dernier dépendent de macros définies dans l'autre

En règle générale, il est très intéressant de pré-analyser les gros fichiers de macros à usage général (c'est-à-dire dont le contenu est stable). Pour les petits fichiers de macros, c'est à vous de voir : sur un Amiga rapide, le gain de vitesse n'est plus vraiment sensible.

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

1.29 Mode préprocesseur

Dans ce mode, Mac2E va remplir son vrai rôle de préprocesseur : il va traiter dans le fichier du champ FROM tous les appels de macros et écrire le résultat dans le fichier du champ TO.

Avec le champ WITH, vous précisez autant de fichiers de macros que vous le désirez. Depuis Mac2E v4.0, ces fichiers peuvent être soit pré-analysés soit "bruts". Mac2E fera automatiquement la différence.

Si un fichier de macros du champ WITH est "brut", il sera intégré à l'ensemble des macros déjà connues de Mac2E, et il sera vérifié qu'il ne comporte pas de macros déjà définies dans un autre fichier de macros chargé avant. Ensuite, les macros qu'il contenait seront pré-analysées seulement en cas de besoin ce qui limite la perte de temps par rapport à un fichier de macros pré-analysé. Lors d'une telle pré-analyse "en direct", les appels de macro internes sont recherchés pour absolument toutes les macros connues au moment de l'analyse du fichier du champ TO. Autrement dit, un fichier de macros "brut" peut dépendre d'un autre fichier de macros (si celui-ci est aussi précisé dans le champ WITH bien sûr).

Si un fichier de macros du champ WITH est pré-analysé, il sera intégré tel quel à l'ensemble des macros déjà connues sans vérifier qu'il contient des macros déjà définies dans un autre fichier de macros chargé avant, ou des macros qui dépendent de macros définies dans un autre fichier de macros précisé ou non dans le champ WITH. En d'autres mots, un fichier de macros pré-analysé ne doit dépendre de rien.

Les drapeaux VERBOSE, KEEPSPACES et DEBUG s'appliquent aussi aux pré-analyses "en directe" que pourrait avoir à effectuer Mac2E en mode préprocesseur.

1.30 VERBOSE

VERBOSE force Mac2E lors d'une pré-analyse du corps d'une macro à afficher le nom de cette macro avant la dite pré-analyse.

Voir les messages d'erreur pour un exemple d'utilisation de ce drapeau.

VERBOSE est incompatible avec DEBUG.

1.31 KEEPSPACES

KEEPSPACES force Mac2E lors d'une pré-analyse du corps d'une macro à conserver dans celui-ci les espaces et les tabulations situés en début de ligne, lorsqu'il se poursuit sur plusieurs lignes. Par défaut, Mac2E les élimine, ce qui diminue la taille du fichier pré-analysé, et accélère le traitement.

1.32 DEBUG

DEBUG force Mac2E lors d'une pré-analyse du corps d'une macro à afficher une description complète de cette macro :

- nom de la macro
- nombre d'arguments
- corps de la macro avant la pré-analyse
- corps de la macro après la pré-analyse

Ce drapeau est présent surtout pour les cas de débuggage (d'où son nom !) où il est intéressant de voir exactement comment Mac2E interprête une définition de macro.

DEBUG est incompatible avec VERBOSE.

1.33 Les messages d'erreur

Tous les messages d'erreur retournés par 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 Mac2E traite les appels de macro internes aux corps d'autres macros pendant une pré-analyse. En effet, cette analyse se fait hors de tout fichier, donc sans numéro de ligne pour indiquer précisément une erreur.
Donc pour connaître l'endroit où se situe une erreur signalée sans numéro de ligne, il faut relancer Mac2E avec l'option VERBOSE. Cela va forcer Mac2E à afficher tous les noms de macro dont il pré-analyse le corps. Ainsi, quand il affiche une erreur, il vous suffit de regarder le nom des macros qu'il vient de pré-analyser, en partant de la plus récente jusqu'à la plus ancienne. Pour chacune (en suivant l'ordre précédent), vous vérifiez si c'est elle qui contenait l'erreur jusqu'à trouver la fautive. Il y a en fait de grandes chances que ce soit la dernière, mais ce n'est pas une obligation. En effet, considérez une macro où dans le corps on trouve successivement un appel correct à une macro, puis un autre appel cette fois incorrect à une autre macro. Lors de la pré-analyse de ce corps, le premier appel va provoquer la pré-analyse de la macro correspondante, puis cela fait, la pré-analyse du corps va se poursuivre jusqu'à l'appel incorrect. Là l'erreur sera signalée, mais la macro fautive sera l'avant-dernière indiquée par VERBOSE car la dernière aura été la macro correspondante au premier appel correct.

Exactement de la même façon, le drapeau VERBOSE est très utile pour déterminer le cycle de dépendance entre macros quand Mac2E détecte un tel cycle.

1.34 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 5 choses :

- Mac2E
- mui.m
- muimaster.m
- mui.ma
- OptiMUI2E

Toute "cette interface" est basée sur MUI v2.2. 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
mui.m
muimaster.m
mui.ma
OptiMUI2E

1.35 mui.m

mui.m est comme son nom l'indique un fichier module 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.

Depuis Mac2E v4.0, toutes les définitions de constantes ont été placées dans ce fichier plutôt que laissées comme macros dans mui.ma. La syntaxe est cependant restée la même, en particulier les parties en minuscules de ces constantes le sont toujours. Donc vos sources n'ont pas besoin d'être adaptés à une exception près : il vous faudra peut-être inclure ce module mui.m dans certains de vos modules (dans le cas de Amiga E v3.0) maintenant, alors qu'avant pré-processer ces derniers avec Mac2E et mui.ma suffisait.

Depuis Mac2E v4.0, mui.m contient aussi la nouvelle constante MUI_TRUE qui a pour valeur un. Cela correspond en fait à la constante TRUE du langage C qui servi à l'écriture de MUI. Donc vous devrez utiliser MUI_TRUE plutôt que TRUE (constante du langage E) avec MUI.

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

1.36 muimaster.m

muimaster.m est comme son nom l'indique un fichier module 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'.

1.37 mui.ma

mui.ma est la clé de voute de "cette interface MUI-Amiga E" puisqu'il contient toutes les macros autres que les constantes (depuis Mac2E v4.0) du fichier mui.h, mais adaptées pour le langage E. La syntaxe des macros de mui.ma, 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.ma 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 Mac2E.

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.ma

Parmi toutes les macros de mui.ma, deux nécessitent des précisions : StringMUI() et set().

La macro String() de MUI a été renommée en StringMUI() car String() correspond déjà à une fonction du langage E.

Si vous avez déjà utilisé la macro set() avec un Mac2E antérieur à la version 4.0, vous avez dû remarquer qu'elle ne marchait pas toujours avec MUI. Une solution était d'écrire à la place :
domethod(objet , [MUIM_Set , attribut , nouvelle_valeur]). Mais ce n'était pas très élègant ! Grâce à Jan Hendrik Schulz qui a découvert l'origine du problème, j'ai pu changer la définition de la macro set() pour qu'elle marche tout le temps. Ce que je viens de dire s'applique aussi à la macro nnset().

1.38 OptiMUI2E

Si vous jetez un coup d'oeil sur mui.ma, 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 très 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.

1.39 Bugs

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

* L'utilisation des fichiers de macros pré-analysés demande quelques
précautions (voir mode pré-analyse et mode préprocesseur).
* Le nombre d'arguments pour une macro est limité à 32.
* Mac2E ne vérifie pas que la limitation précédente est respectée.

1.40 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
Version 4.0 :  - complètement reprogrammée pour Amiga E v3.0 avec utilisation
                   des options d'optimisation, de la méthode de "hashing"
                   fournie avec Amiga E v3.0, de la programmation OO, de
                   nouveaux algorithmes très rapides pour la manipulation des
                   chaînes de caractères et une meilleure (et surtout plus
                   rapide) gestion des entrées/sorties
               - les pré-analyses sont environ 5000000001lus rapides et les phases
                   de "préprocessing" environ 2 fois plus rapides qu'avec
                   Mac2E v3.x
               - PreMac2E a été fusionné à Mac2E qui est donc maintenant le
                   seul exécutable (voir appel de Mac2E pour sa nouvelle
                   syntaxe d'appel)
               - on peut donner à Mac2E des fichiers de macros pré-analysés
                   ou non, et dans ce dernier cas, il fera la pré-analyse
                   selon les besoins "en direct" (voir mode préprocesseur )
               - possibilité de mettre des commentaires dans le corps des
                   macros, même dans le cas où ce corps s'étend sur plusieurs
```

```
    lignes (voir  des commentaires dans le corps )
- possibilité d'étendre un appel de macro sur plusieurs lignes
    (voir  traiter le passage d'arguments )
- possibilité de donner une liste E comme paramètre (voir
     caractères spéciaux )
- Mac2E peut être interrompu à tout instant par un Ctrl-C
- ajout d'un mode "DEBUG" à Mac2E (voir  DEBUG )
- Mac2E vérifie les doubles déclarations de macros
- Mac2E détecte les cycles de dépendance entre macros
- Mac2E n'a plus de limitation quant à la taille des noms et
    des corps des macros
- Mac2E rapporte correctement les erreurs d'écriture
- quelques bugs mineurs corrigés
- Mac2E est maintenant GiftWare ! (voir  distribution )
- OptiMUI2E v1.2 a été réécrit plus proprement pour Amiga E
    v3.0
- mise à jour de mui.ma par rapport à MUI v2.2
- les définitions de constantes ont été déplacées dans mui.m
    (voir  mui.m )
- nouvelle constante MUI_TRUE introduite (voir  mui.m )
- deux macros intéressantes à regarder (voir  mui.ma )
```

1.41 Futur

```
    Pour Mac2E v3.0, j'écrivais :
<<
    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 !
>>
    Donc maintenant, je n'attends plus que vos suggestions :-).
```

1.42 Distribution

```
    Tous les fichiers de cette distribution restent sous copyright de
l'auteur (Lionel Vintenat). Vous n'êtes donc autorisé à les modifier que
pour votre usage STRICTEMENT PERSONNEL.
    Les seules exceptions sont les fichiers "mui.m", "muimaster.m", "mui.ma",
"Readme.mui" et les icônes. Vous pouvez aussi bien sûr vous servir des
sources fournis dans cette archive pour vos propres programmes : ils
sont là pour ça !


    Cette archive peut être librement distribuée par tous les moyens
imaginables (serveur ftp, BBS, collection du domaine publique, etc), tant
que les deux conditions suivantes sont respectées :

    1) Personne ne tire AUCUN bénéfice de cette distribution. En parti-
culier, si Mac2E est diffusé sur une disquette, celle-ci ne doit pas être
vendue pour plus de 4$ US (ou équivalent), et s'il s'agit d'un CDROM,
celui-ci ne doit pas être vendu pour plus de 30$ US (ou équivalent). Tout
autre type de vente (avec profit) ne peut EN AUCUN CAS être effectué sans
l'autorisation de l'auteur. Les seules exceptions sont les CDROM de Fred
```

Fish et d'aminet, qui eux (et seulement eux !) peuvent inclure Mac2E dans leur collection sans me demander la permission. En particulier, cela exclut DEFINITIVEMENT à France Festival Distribution le droit de distribuer Mac2E (j'insiste LOURDEMENT sur ce point...). Mais sans doute cela ne génera-t-il pas vraiment Mr Serge Hammouche, qui n'hésite pas à traiter (ouvertement sur certains réseaux français) d'incapables les programmeurs de DP français...

2) Cette archive est distribuée DANS SON ENTIER, et SANS MODIFICATION par rapport à sa version originale sur aminet. Cela signifie en particulier que si vous faites une traduction de la documentation ou du catalogue dans un nouveau langage, ou que vous corrigez des bugs et recompilez l'exécutable, vous DEVEZ me les faire parvenir pour que ce soit MOI EVENTUELLEMENT (surement en fait s'il s'agit de traductions) qui redistribue une nouvelle version de ce programme. La structure que doit avoir cette archive est indiquée dans le fichier "LisezMoi.d_abord" de cette distribution.

Toute distribution de Mac2E qui ne respecte pas les deux conditions précedentes sans mon autorisation est ILLEGALE.

Tout le début de ce paragraphe peut sembler très strict, à la limite de la paranoïa même, mais étant donné les pratiques douteuses de gens comme Serge Hammouche qui vend à des prix astronomiques (sous prétexte de traduction) des logiciels librement distribuables sans même en avertir les auteurs, il me semble nécessaire de protéger mes droits. Je fais des programmes pour mon plaisir sans aucune prétention pour gagner de l'argent, et je suis heureux si ils peuvent aussi aider d'autres personnes, mais que des gens en profitent pour se faire de l'argent sur mon dos : NON ! Les limitations précédentes ne visent absolument pas les gens sérieux comme Fred Fish, le Club Amiga Montréal, les systèmes opérateurs d'aminet, ou tous les clubs de passionnés, qui eux supportent réellement le domaine publique sur Amiga. Elles visent uniquement les organisations aux pratiques discutables comme France Festival Distribution.

D'autre part, je dégage toute responsabilité quant à l'utilisation de ce programme et les dommages directs ou indirects qu'il pourrait causer. Que ce soit clair : VOUS L'UTILISEZ A VOS RISQUES ET PERILS !
Cependant, je pense l'avoir suffisament testé et fait testé pour dire qu'il ne comporte plus de bugs sérieux.

Enfin, ce programme est distribué selon le concept Giftware. Autrement dit, vous devez m'envoyer un cadeau si vous utilisez Mac2E ! :)
En effet, je fais des programmes par plaisir et par besoin. Mon but n'est certainement pas de gagner de l'argent avec. Cependant distribuer un programme demande pas mal de travail supplémentaire (documentations, script installeur, etc), aussi j'aimerais recevoir un retour de ceux qui utiliseront Mac2E. En fait, tout signe de vie sera TRES apprécié, même un simple email ou une carte postale. Ce qui m'interesse avant tout, c'est le contact avec d'autres personnes. Mais pour vous aider à choisir mon cadeau, voici quelques suggestions :) :

- un enregistrement (gratuit !) à un programme shareware
- une de vos réalisations (programme, module, animation, image, etc) si elle n'est pas facilement récupérable sur aminet

- des sources E, C ou assembleur qui touchent de près la programmation système, ou encore des sources False ou BrainFuck (ils peuvent bien toucher ce qu'ils veulent, du moment qu'ils marchent ! :))
- de l'argent, hummmm, pourquoi pas ? :)
- vos vieux RKM 1.3 (ou mieux 2.0)

J'insiste sur le fait qu'il est très frustant de faire l'effort de placer son programme dans le domaine du librement distribuable sans jamais recevoir de retour, c'est tout juste si vous savez qu'on utilise votre programme ! Alors SVP supportez le concept GiftWare, tout le monde s'y retrouve : l'auteur est heureux de recevoir un retour, et ça ne coute quasiment rien aux utilisateurs.

1.43 L'auteur

Vous pouvez me joindre par courrier à mon adresse familiale :

Lionel Vintenat
3 impasse Boileau
Lotissement Les Termes
87270 COUZEIX
FRANCE

Vous pouvez également me joindre sur internet. Mon adresse e-mail est vintenat@irit.fr. Celle-ci restera très vraisemblablement valable jusqu'en septembre 1994 inclus, mais je n'y aurai pas accés pour cause vacances/service militaire avant la fin de ce fameux mois de septembre. Alors ne comptez pas trop sur une réponse rapide si vous utilisez cette adresse ! Si vous tenez vraiment à me joindre d'ici là, mieux vaut m'écrire directement à mon adresse familiale (n'oubliez pas de le faire avec une jolie carte postale de votre pays 8)).

Mais la meilleure solution est quand même à mon avis d'attendre que je redonne signe de vie sur le réseau en septembre 1994, car j'aurai à ce moment là une nouvelle adresse internet valable pour toute la nouvelle année scolaire.

1.44 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 : si vous voyez des phrases étranges dans cette documentation, ce n'est pas sa faute, cela vient probablement de mes mises à jour personnelles :-)
- 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 qui m'ont envoyé des rapports de bugs et des encouragements, et parmi eux plus spécialement Jan Hendrik Schulz qui m'a donné

beaucoup de très bonnes idées pour l'interface entre MUI et Amiga E (notament la solution du problème de la macro set())
- 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 !

1.45 Index