A only tree

Nous avons choisi de générer un seul arbre avec tout le code figurant dans le fichier kooc, pour que les dépendances de la CNORM comme les références vers les types ne soient pas corrompu.

On conserve en parallèle un deuxième arbre qui lui va stocker les références vers les membres de nos modules, packages et classes afin que les appels kooc puissent faire appel au bon élément et tester si les éléments existent réellement.

Exemple :

* project
  + block
    - [0], [1], [2]
  + kooc
    - modules
      * [« Test »], [« Toto »]
    - packages
      * [« Pkg1 »]
    - Classes
      * [« Test1 »]

Typedef, Struct, Union a gogo

La CNORM est loin d’être parfaite, du moins, quand on l’utilise pour le kooc, imaginons le code suivant :

(1) typedef int Entier ;

@module A

{

(2) typedef int Entier;

(3) Entier a ;

}

A l’aide de la règle déclaration, la CNORM va dans (1), ajouter une référence dans le nœud « types » de l’arbre avec comme key « Entier ».

Puis arrive à la ligne (2), la CNORM va voir que dans le nœud « types » il existe déjà un élément appelé « Entier », donc ERROR !

Il faut trouver un moyen pour différencier les deux types, pour cela on essaye en manglant les types dans les modules, l’ERROR existe encore, PK ? Car nous pouvons mangle que après que la CNORM est ajoute dans l’arbre le type mais comme il existe déjà…ERROR !

De plus si on regarde la ligne (3), si on arrive a mangle le type de la ligne (2), on a un autre problème a regle, celui de l’appel a ce type, « Entier » lui sera lie au type définie a la ligne (1) et non celui a la ligne (2) .

Solution :

De nouveaux mots-clés peuvent régler ce problème, un petit exemple pour illustrer cela :

(1) typedef int Entier ;

@module A

{

(2) @typedef int Entier;

(3) @typeof Entier a ;

}

Si on se trouve dans des blocks Kooc, on utilise des mots-clés commencent par « @ » pour différencier explicitement la déclaration C du code Kooc et ainsi de mangle avance de stocker l’élément dans l’arbre, puis a la ligne (3) en utilisant « @typeof » on refait appel au type mais cette moi nous faisons appel au type mangle a la ligne (2).

TODO

* Envoyer ERROR si prototype dans [@module|@class] mais pas de fonction dans @implementation
* Reparer probleme @class List {@member List \*next ;} qui ne marchera pas puisque au moment ou le C compile dans la structure List le typedef portant le nom List n’existe pas.
* Créer une fonction generique qui cree une backup d’un fichier portant le meme que le fichier qu’on souhaite generer avec pour extension ‘.bak’.
* Créer une fonction generique qui delete les fichiers en cas d’erreur rencontrer lors la koocatation.
* Créer la classe virtuel Object pour manipuler les classes provenant de package.
* Trouver solution pour la compilation de fichier multiple

DONE

* @module Test
  + genere km\_test.c, km\_test.h
* @class Test
  + genere kc\_test.c, kc\_test.h, kc\_test.meta.h, kc\_test.meta.h
  + @member [{}]? mangle sinon non mangle en static,
  + reconnaissance du member si contient en parametre le self
  + gere l’heritage simple, si attribut dans parent genere un .parent.attribut
* @implementation Test

@ IMPORT

@ MODULE

@ CLASS

@class Variant

{

@member

{

void init(int) ;

void init(char) ;

void init(char\*) ;

void clean() ;

}

@member int toInt() ;

@member char toChar() ;

@member char\* toCStr() ;

void getVersion();

@member int \_integer;

@member char \_character;

@member char\* \_cstr;

}

Output  .H:

typedef struct ~~Variant~~

{

int ~~\_integer~~

char ~~\_character~~

char\* ~~\_cstr~~

} Variant ;

Output .C:

static void getVersion()

{

printf(“0.42”);

}

void ~~init~~(int)

{}

void ~~init~~(char)

{}

void ~~init~~(char\*)

{}

void ~~clean~~()

{}

…

@ IMPLEMENTATION

@ PACKAGE

@package A

@package A.B

@package A.B.C

@package A

{

@package B

{

@package C

{

}

}

}