M2 - Compilation Avancée

(CPA 2016-2017)

TD6

Création d'une nouvelle directive avec plugin GCC

hugo.brunie.ocre@cea.fr
julien.jaeger@cea.fr
patrick.carribault@cea.fr

I Déclaration d'une simple directive

Le but de cette partie est de récupérer les informations contenues dans la directive à partir d'un plugin GCC.

Q.1: En vous aidant de la documentation sur les plugins GCC, déclarer une nouvelle directive. Cette directive servira à définir le nom d'une fonction sur laquelle nous souhaiterons appliquer un traitement spécial à partir d'une nouvelle passe. Dans un premier temps et à partir de la fonction passée à la fonction *c_register_pragma*, affichez uniquement un message lorsque le nouveau pragma est reconnu.

Cette directive sera utilisée de la manière suivante dans le code qui l'exploitera :

```
#pragma instrument function ma_fonction
```

Correction

Voir fichier **plugin_TP56_1.cpp**.

Q.2: En vous inspirant de la fonction *handle_pragma_target* présent dans le fichier `c-family/c-pragma.c`, afficher uniquement le nom de la fonction annotée lorsque l'argument du pragma est de type `CPP_NAME`.

Correction

Voir fichier **plugin_TP56_2.cpp**.

Q.3: Étender maintenant cette directive pour gérer plusieurs noms de fonction. Toujours en vous inspirant de la fonction *handle_pragma_target*, parser et afficher ces noms.

Cette directive pourrait être utilisée de la manière suivante dans le code qui l'exploitera :

```
#pragma instrument function(f1,f2,f3,f4)
```

Correction

On commence par vérifier si le premier token est une parenthèse ouvrante. Ensuite on affiche le premier nom. Si on a rencontré une parenthèse ouvrante, alors il peut y avoir une liste de nom : on lit les virgules de séparations. Une fois la liste finie, on doit trouver une parenthèse fermante. On renvoie un message d'erreur si le format n'est pas valide (`#pragma instrumente function NAME` ou `#pragma instrumente function (NAME1,NAME2,...)`). Voir fichier **plugin_TP6_3.cpp**.

Q.4: Toujours en vous inspirant du fichier `c-family/c-pragma.c`, rajouter un test dans le traitement de la directive, afin de vérifier qu'elle a bien été placée à l'extérieur d'une fonction. Retournez une erreur dans le cas contraire.

Correction

Voir fichier **plugin_TP56_4.cpp**.

II Interaction avec une passe

Nous souhaitons maintenant que notre directive puisse interagir avec une passe. Pour ce faire, nous allons enregistrer les noms des fonctions passées par la directive dans une liste définie en variable globale du plugin.

Q.5: Construire cette liste en utilisant la structure vecteur définie dans le fichier `vec.h` des sources GCC. Afficher le contenu de ce vecteur après chaque pragma.

Correction

Déclaration d'un vecteur global de `char *`. Utilisation de *safe_push* pour insérer les noms dans le vecteur. Voir fichier **plugin_TP56_5.cpp**.

Q.6: Pour le moment, certains noms présents dans des pragmas non-valides sont insérés dans la liste. N'insérer que les noms présents dans les pragmas valides. Vérifier qu'un nom de fonction n'a pas été utilisé plus d'une fois dans la directive. Si c'est le cas, afficher un warning.

Correction

Déclaration d'un vecteur temporaire de `char *`. On ne reverse les noms de ce vecteur dans le vecteur global que si le pragma a été validé. On profite de la copie pour vérifier que le nom de la fonction n'est pas déjà prsente dans le vecteur. Voir fichier **plugin_TP56_6.cpp**.

Q.7: En vous aidant des TDs précédents, créer une passe qui affiche le nom de chaque fonction. Modifiez la gate pour que cette passe soit activée seulement si le nom de la fonction est contenue dans la liste globale.

Correction

On reprend ce qu'on a fait dans le TD2 q3. Seul changement : la fonction gate ne renvoie plus `true`. On récupère le nom de la fonction courante sour forme de `char *`, et on renvoie la valeur de la fonction *already_pushed* de la question précédente. Voir fichier **plugin_TP56_7.cpp**.

Q.8: En quittant GCC, vérifiez que tous les noms de fonction contenus dans la liste globale ont été utilisés. Dans le cas contraire, affichez un warning sur les noms restants. Il s'agit probablement de fonctions non-définies dans le code annoté.

Correction

On enregistre un nouveau callback avec en argument de position `PLUGIN_FINISH` et la fonction *plugin_finalize*. On modifie la gate pour enlever du vecteur les noms de fonctions trouvées à la compilation. Dans la fonction *plugin_finalize*, on vérifie la taille du vecteur global. Si il n'est pas nul, alors il reste des arguments que l'on a pas rencontré. On les affiche. Voir fichier **plugin_TP56_8.cpp**.