

# TD – E-ACSL

## Rendu

Le but est de rendre un rapport présentant vos réponses aux questions de ce TD. Vous devez me rendre le rapport et **uniquement** le rapport. Il doit donc présenter **tous les éléments nécessaires** pour démontrer que vous avez bien compris la question et bien répondu à la question (et donc être illustré par des screenshots des éléments importants, par des copier/coller des morceaux de code importants, etc). **Mettez l'accent sur vos explications**, ce sont elles qui importent le plus. Le résultat n'a de valeur qu'en lien avec son côté explicable (comme lors d'un audit réel en fait).

Le rapport doit être dans l'un de ces formats :

- PDF (préféablement)
- DOCX (et surtout PAS DOC)
- ODT

Format du nom de fichier : td-e-acsl-Nom1-Nom2.pdf

Envoyez votre rapport à l'adresse [allan.blanchard@cea.fr](mailto:allan.blanchard@cea.fr) au plus tard le **07/03** avec comme objet :

[INSA CVL] E-ACSL

## 1 Prise en main d'E-ACSL

```
#include <limits.h>

int add(int* x, int *y) {
    //@ assert INT_MIN <= *x + *y <= INT_MAX ;
    return *x + *y ;
}

int main(void) {
    int a = 4 ;
    int b = 2 ;
    int r = add(&a, &b) ;
    //@ assert r == 6 ;
}
```

1. Construisez un programme instrumenté avec la commande :

```
e-acsl-gcc.sh simple.c -c -O exe
```

Dans le fichier `a.out.frama.c`, identifiez les instrumentations correspondant à l'annotation, pouvez-vous expliquer chacune d'entre elles? Comment est géré le débordement possible des entiers?

2. Le programme en question n'a pas d'erreur, vérifiez qu'aucune erreur n'est levée par l'exécutable `exe.e-acsl`. Modifiez le programme pour introduire une erreur qui invalide l'assertion d'origine et constatez que l'erreur est bien détectée.

3. Pour chaque autre problème potentiel construisez un programme qui le déclenche et comparez l'exécution avec et sans instrumentation. (Limitez vous à un seul débordement entier).

## 2 Une première spécification

```
int abs(int x) {
    if(x < 0) return -x ;
    else return x ;
}

int main(void) {
    // This syntax says that we cast the result in "void":
    // we purposely drop the result in the code
    (void) abs(-1);
}
```

1. Spécifiez la fonction abs en ACSL
2. Ajoutez des appels supplémentaires dans la fonction main et vérifiez au fur et à mesure que la fonction fait son travail en compilant avec E-ACSL :

```
e-acsl-gcc.sh abs.c -c -O exe
```

et en exécutant les exécutables produits.

3. Écrivez une version erronée de la fonction abs avec la même spécification et vérifiez qu'E-ACSL détecte bien le problème.

## 3 Absence d'erreurs à l'exécution

```
int sum(char *a, unsigned size){
    int sum = 0;
    for(int i = 0 ; i <= size ; ++i){
        sum += a[i];
    }
    return sum;
}

int main(void){
    char a[3] = { 1, 2, 3 };
    return sum(a, sizeof(a));
}
```

1. Construisez un programme instrumenté avec la commande :

```
e-acsl-gcc.sh rte.c -c -O exe
```

puis exécutez l'exécutable normal (./exe) et puis celui instrumenté (./exe.e-acsl),

2. Ajoutons la vérification d'absence d'erreurs à l'exécution :

```
e-acsl-gcc.sh rte.c -c --rte=all -O exe
```

puis exécutez l'exécutable normal (./exe) et puis celui instrumenté (./exe.e-acsl),

3. Que constatez-vous ? Qu'en déduisez-vous à propos d'un programme C qui "s'exécute correctement" ? Corrigez le programme et vérifiez que l'erreur n'est plus levée.

## 4 Trouver la dernière valeur

La fonction suivante :

```
#ifndef __FIND_LAST_OF
#define __FIND_LAST_OF

#include <limits.h>

int find_last_of(int const* a, int len, int value);

#endif
```

doit trouver l'indice de la dernière occurrence de `value` de la séquence d'au moins `len` éléments pointée par `a`. Si l'élément n'est pas présent, elle retourne `INT_MAX`.

1. Dans `find_last_of.h`, donnez une spécification ACSL à la fonction,
2. Ajoutez des tests pour cette fonction dans le fichier `find_last_of_main.c`, pour compiler avec l'instrumentation, lancez :

```
e-acsl-gcc.sh find_last_of.c find_last_of_main.c --rte=all -c \
-O exe
```

3. Il semble que la fonction ait un problème ... (Non ? Retournez à la question 2). Corrigez la fonction et assurez-vous de sa correction.

## 5 Personnalisons nos erreurs

Reprenons notre premier exemple et essayons de classifier nos erreurs. L'idée est de fournir un fichier supplémentaire `my_assert.c` contenant une implémentation différente pour la fonction d'assertion E-ACSL. Le squelette du code vous est fourni.

Si une assertion est :

- un souci d'accès mémoire, son nom commence par : `mem_access`, ou `rte/mem_access`,
- un souci de débordement entier, son nom commence par : `rte/signed_overflow`

La fonction `strstr` pourrait être utile ...

1. Construire une fonction qui :
  - indique une erreur critique en cas d'accès mémoire et abort le programme,
  - indique une erreur sérieuse en cas de débordement entier et exit le programme,
  - indique les autres erreurs sans arrêter le programme.

Pour compiler votre programme, lancez :

```
e-acsl-gcc.sh my_test_file.c -c -O exe --rte=all \
--external-assert=my_assert.c
```

2. Trouvez des programmes de test déclenchant chacun des cas de votre fonction. Comparez et commentez les résultats d'exécution avec ou sans instrumentation du programme.