Documentation utilisateur

Usage général :

`$./proj.run <list of prog file paths> <verboseLevel(0~5)>`

Si on ne fait rien, les mini-programmes sont exécutés.

Pour lancer le mode débugger : envoie de SIGQUIT (ctrl+\),

Le processus débogué est celui dont l'instruction était en cours lors de l'envoi du signal.

Commandes du mode débugger :

`continue`

Continue les instructions du processus en cours de débogage.

`step`

Avance le processus d'une instruction.

`print <variable>`

Affiche le contenu de la variable `<variable>`.

`display <variable>`

Affiche le contenu de la variable `<variable>`,

Et, lors des `step`s suivants, affiche toutes les variables dont `display` a fait l'objet.

`show <quoi>`

`<quoi>` peut être "`display`", "`break`" ou "`proc`"

Affiche respectivement les variables à afficher lors des `step`s, les points d'arrêts spécifiés par la commande "`break`", les processus traçables.

`remove <quoi> <n°>`

`<quoi>` peut être "`display`" ou "`break`"

Supprime la variable à afficher, respectivement le point d'arrêt, qui porte le numéro `<n°>`

(pour avoir les numéros, utiliser "`show`").

`stop`

Arrête le processus courant.

`start <step | >`

Démarre le processus courant.

Si `step` est spécifié, le processus est démarré et une seule instruction est exécutée.

`restart <step | >`

Redémarre le processus courant (équivaut à `start` puis `stop`).

`status`

Affiche le statut du processus courant.

`changeproc <n°>`

Change le processus débogué en celui qui a le numéro spécifié.

`quit`

Termine tous les processus puis programme.

(note: si un processus n'a pas été tracé, il n'y aura pas d'affichage comme quoi il est terminé)

`end`

Termine le débugger, reviens à l'exécution normale (sans le débugger).

Si il reste des instructions à exécuter, elles le seront.

Extension du mini-langage : les signaux

Dans le corps du mini-programme, on écrit UN traitant, on peux ajouter des signaux à dérouter, et en enlever. Tous les signaux déroutés vont exécuter le même traitant.

/!\ L'interruption d'un traitant avec SIGQUIT le fait se terminer.

C'est à dire que si on envoie SIGQUIT pendant un traitant, le prompt apparaît, mais si on décide de reprendre le processus (`continue` ou `step`), on reprendra à l'exécution qui suit celle où on a envoyer SIGQUIT pour interrompre le processus.

Quatre primitives ont été ajoutés au mini-langage :

SIGNAL

ENDSIGNAL

SIGADD

SIGDEL

SIGNAL s'utilise avec ENDSIGNAL, entre ses deux primitives, on met les instructions à exécuter lors d'un déroutement de signal (c'est le traitant).

Ce bloc doit être compris entre PROGRAM et ENDPROGRAM, les variables déclarées avant le bloc sont invisibles.

SIGADD équivaut à un déroutement, il s'utilise ainsi :

```
SIGADD @ <int>
```

où `<int>` est le numéro du signal à dérouter (utiliser des chiffres, pas SIG*)

SIGDEL vaut l'annulation d'un déroutement

Il s'utilise comme SIGADD.

Si dans le mini-programme, on trouve un SIGADD, mais pas de traitants (et vice-versa), c'est comme si on avait rien fait.

Modification par rapport au mini-langage initial :

L'instruction READ a été est « protégée », on ne peux taper qu'un entier (sinon, un message d'erreur apparaît et vous invite à entrer de nouveau un entier).

Si, dans le mini-programme, on a plusieurs instructions READ à la suite, on ne peux pas taper les entiers sur une même ligne, il faut presser Entrer.
Par exemple, si deux instructions READ se suivent, l'input « 4 5 » exécute la première instruction avec 4 comme input, le 5 est ignoré, il faut donc maintenant taper l'entier qui sera pris en compte pour la deuxième instruction.