**Introduction**

Le sysRPL est un langage intermédiaire entre l'assembleur et l'userRPL (langage de l’utilisateur des calculatrice hp), dans laquelle est programmé le système de la calculatrice. C'est un compromis entre langage de haut niveau et plus rapide que l'userRPL. L'assembleur restant toujours le plus rapide langage.

Il utilise la notation polonaise inversé (RPN).

**I La notation polonaise inversé (RPN)**

Dans tout le guide il est supposé que la calculatrice est réglée sur le mode RPN, par opposition au mode algébrique.

Pour passer de l'un à l'autre, il suffi de presser la touche **[MODE]** puis sur "CHOOSE" (touche **[F2]**).

Vu que ce mode est essentiel pour la programmation en sysRPL, je vais rapidement expliquer ce mode de saisie des commandes.

Essayions de calculer 2+3. Le résultat est connue, 5, mais pour cela suivre mes instruction.

Tout d'abord taper 2 **[ENTER]** 3  **[ENTER]**. Sur les deux lignes du bas vous avez:

2: 2

1: 3

Cela signifie que vous avez 2 au niveau 2 de la pile et 3 au niveau 1.

Qu'est-ce la pile? C'est quelque chose de fondamentale pour la notation RPN.

Il s’agit d’un espace de mémoire sur lesquelles les opérations sont faites. Par exemple l'addition (la soustraction, la division, la multiplication...) demande deux arguments, l'appui sur la touche [+] va additionner les deux premiers niveaux de la pile.

Essayiez. Vous obtenez 5 au niveau 1 de la pile. Réessayer, un message d'erreur apparait, il n'y a rien au niveau 2 de la pile.

Appuyiez deux fois sur **[ENTER]**, vous obtenez 5 aux trois premiers niveaux de la pile.

La touche **[ENTER]** est un raccourcit vers la commande *DUP* qui copie le niveau 1 de la pile. Appuyiez sur **[+]**, vous avez 10 au niveau 1 et 5 au niveau 2.

Quelques règles générales:

\* La pile n'est limité que par la mémoire, seul 7 niveau sont visible par défaut, mais un appui sur la flèche du haut permet de les voir tous.

\* Lorsqu'on pose un élément sur la pile, il passe au niveau un de la pile, l'élément au niveau 1 va niveau 2...

\* Toutes les opérations se font sur la pile.

\* Un appui sur une touche, comme **[+]**, **[UPDIR]**, un répertoire, va tout de suite exécuter la commande.

(Contrairement au mode algébrique ou il faut presser **[ENTER]**)

\*Il existe des commandes spécifiques pour manipuler la pile. DUP, en userRPL, permet de copier le niveau 1 de la pile, SWAP inverse les niveaux 1 et 2…

Pourquoi un système pouvant paraitre un peu compliqué?

\* Historiquement, c'est plus facile à programmer.

\* Ensuite, par comptabilité, les habituer ayant du mal à s'en passer.

\* Ce mode économise des parenthèses, parfois illisible sur certaine calculatrices.

Par exemple, pour taper (2+x) / (y-3) on tape: *2 x + y 3 - /*

Ce mode est à connaitre si on veut programmer en sysRPL. En effet, les instructions sont en RPN.

**II Essai sur la calculatrice, compréhension du sysRPL**

Le sysRPL (Revish Polonaise Language système - par opposition au userRPL, soit le langage habituelle en mode RPN - est un langage intermédiaire entre l'assembleur et le langage utilisateur.

Pour programmer dans ce langage il faut utiliser la bibliothèque 256, intégré dans la calculatrice, MASD, qui sert aussi à programmer en assembleur Saturn et ARM.

Pour attaché une bibliothèque il faut la redémarrer en appuyant simultanément sur **[ON]** et **[F3]**.

Taper alors *256 MENU* **[ENTER]**, vous avez accès à cette bibliothèque.

La documentation peut être téléchargée sur le site hp-calc.org à cette adresse:

<http://www.hpcalc.org/hp49/docs/programming/masddocs.zip>

Une version plus ancienne a été (mal) traduite en français sur le site. Tous la partie ARM n'y figure donc pas. (Note : librery se traduit par bibliothèque pas par librery)

Taper *{ + } 1 GET* **[ENTER]** , vous déposer la commande *+* sur la pile. Voilà une méthode pour déposer une commande sur la pile sans l'évaluer. Pour l'évaluer vous pouvez utiliser **[EVAL]**.

Avec *+* au niveau 1 de la pile, tapé *COMP->* (où "->" le signe shift-droit + **[0]** de la hp).

Vous obtenez 53 au niveau 1 de la pile. La pile a 54 niveaux, la plupart remplit avec des caractères "external". Dès que vous voyez ce mot, vous avez affaire à du sysRPL, et alors soyez prudent, vous risquez le "Try to recovery memory". Traduction (libre): mémoire effacé.

Faite *->PRG*  , vous obtenez le code en sysRPL de *+*. Vous pouvez mettre 1 aux niveaux 2 et 3, puis presser **[EVAL**], vous obtenez 2.

De manière générale, l’userRPL est codé en sysRPL qui est codé en assembleur.

Le sysRPL est semblable au userRPL mais avec deux grandes différences:

-Les instructions ne sont pas systématiquement testées, il y a des commandes spécifiques pour le faire. Conséquence: si il ni a pas d'argument sur la pile et que vous faites une addition, vous risquez de perde vos donné.

-Il y a des instructions spécifiques pour chaque types d'arguments (Entiers, décimaux, entiers système, chaine de caractère, GROB pour les dessins...). Il y a ainsi plusieurs "+", "-"...

Une instruction en userRPL est en faite une d'instructions en sysRPL cohérente pour avoir une version de la commande "+" sur et polyvalente.

A partir du code en sysRPL au niveau 1 de la pile, on obtient le code source avec la commande *->S2*.

Si vous faite maintenant avec la commande "+" en sysRPL on obtient un résultat un peu illisible.

A peu près:

"!NO CODE

!RPL

::

PTR 26305

# 11

PTR 3035F

... (Je saute)

;

@"

Cependant si vous utilisez la commande ASM, vous obtenez de ce fichier le programme en sysRPL.

Dans la partit III, nous allons expliquer comment rendre plus lisible. Pour l'instant décryptons juste un peu le programme.

**!RPL** indique le mode sysRPL, **!ASM** pour programmer en assembleur Saturn, et **!ARM** pour l'assembleur ARM.

**!NO CODE**, pour ça, j'avoue mon ignorance. Il me semble qu'il faut le mettre pour le sysRPL, pas pour l'assembleur.

:: commence un bloc, et ; le termine, en sysRPL, indispensable.

@ conclue le programme. Le caractère # indique un entier système, un type d'objet spécifique à la programmation en sysRPL. Le caractère Z désigne un entier classique, % pour les décimaux. Un nombre non précédé de caractère est, à priori entier système.

Pour les commentaires, il y a deux possibilité: soit les parenthèses, comme :

( Ceci est un commentaire).

Ce commentaire peut être multilingues, soit un signe "\*" en début de ligne.

\* Ceci est un commentaire.

\* Il finit à la fin de la ligne.

**III Pour programmer sérieusement**

Il est possible de programmer avec des PTR 26305, mais se n'est pas pratique à retenir. Il est possible de rajouter une bibliothèque qui permet d'avoir des noms de commandes comme SWAP, %+, #\*... plus humainement compréhensible.

Comme je suis fainéant, je vous proposerai un programme qui offre une documentation sur les commandes. Mon rôle pourra se contenter de présenter le programme et donner des règles de syntaxe.

Le programme s'appelle Emacs comme le bien connu éditeur de texte Unix.

Il est téléchargeable sur hpcalc.org à l'adresse suivante:

<http://www.hpcalc.org/hp49/apps/editors/emacs211a.zip>

**3.1 Installation d' Emacs**

*3.1.a Installation d’une bibliothèque – règles générales*  (Pour eux qui l'ignore)

Pour installer une bibliothèque mettez au niveau 2 de la pile la bibliothèque et au niveau 1 le port où vous voulez l'installer, puis utilisez la commende STO.

Il y a 3 ports (espace mémoire autre que HOME et dépendance): le port 0 partage le même espace mémoire que HOME, si vous plantez la calculatrice et perdez les données de HOME vous perdez ceux qu'il y a dans ce port. Le port 1 est en RAM. Le plus grand, le port 2 est en FLASH. De préférence choisissez le port 2.

Une fois qu'une bibliothèque est dans un port, elle doit être attaché en la redémarrant la calculatrice avec pressant simultanément [ON] et [F3].

Si une bibliothèque bloque le démarrage, (cas d'une bibliothèque (personnelle :-) mal faites) la touche [DEL] (la flèche <- pas la flache gauche) permet de ne pas attacher les bibliothèques.

*3.1.b Installer Emacs*

Pour installer Emacs c'est simple.

Si vous avez une hp4g installez la bibliothèque EMACS.49G . Pour les autres modèles (hp50g, hp48gII, hp49g+) prenez la version EMACS.49GP . C'est l'éditeur de texte proprement dit.

Pour la doc, si vous ne voulez pas ou ne pouvez pas utiliser la carte SD (modèles hp49g et hp48gII) installez la bibliothèque SDIAG.49G. Vous n'avez qu'un résumé de la documentation possible.

Si vous souhaitez utilisez la carte SD, (recommandé) installez la bibliothèque SDIAG.49GP sur votre calculatrice et copier le répertoire SDIAG sur la racine de votre carte SD.

Pour remplacer les PTR xyz par des commandes lisible prenez la bibliothèque external2.lib.

Une bibliothèque, nosy, téléchargeable sur hpcalc.org permet de d'obtenir le programme source des commandes et peut interagir avec emacs.

**3.2 Présentation d'emacs**

Après avoir tout installer, appuyiez sur **[APPS]** et choisissez emacs.

Vous avez trois options, choisissez la premier (la touche **[F1]**); vous entrez dans l'éditeur emacs.

Par défaut, sur l’éditeur, vous avez déjà du texte déjà prêt pour le sysRPL.

Avec la touche **[NXT]** vous avez en tout 18 options.

Les plus essentiel, (soit celle que je connais), sont:

- CO.. : Complète si commende existante. Si vous tapez SW puis pressez la commande vous avez un choix de possibilités commençant par SWAP. Tapez XYG vous voyez tout de suite XYGROBDISP qui s'affiche.

- ( -> ) (**[NXT]** puis **[F4]**) : Tapez une commande puis le curseur dessus choisissez cette option. Vous avez un résumer de la commande (en anglais). La carte SD permet d'avoir plus d'info.

Prenons l'exemple de la commande CK&DISPATCH0 (tapez CK puis choisissez CO..) vous avez une explication de la commande avec cette option. La touche [F4] permet de voir des commandes liées. Si vous avez installé nosy le code source est visible avec [F3].

Si au lieu de taper juste sur ( -> ), vous faite shift-droit + ( -> ), vous accédé à la liste des commandes en sysRPL classé par catégories.

-ENDOB permet, si vous avez nosy de consulter le code source d'une commande.

Vous savez maintenant, où chercher de l'aide.

Si on appui deux fois sur **[NXT]** on obtient une série de structure utile.

1) La première, permet d'obtenir des bloc qui commence par :: et finissent par ; .

L'utilisation de shift-droit/gauche permet des variantes.

2) La touche 2 permet de définir des variables locales, valables dans un environnement local, par opposition à une variable globale qui est définit pour toute la calculatrice.

Une variable locale est introduite par LAM, comme LAM MaVariable.

Une variable globale est introduite par ID comme ID CASDIR.

Il existe des variables locales sans nom, qui me semble difficile à utiliser.

Il existe aussi des variable converties en variables sans nom à la compilation. Elles n'ont pas de prépositions.

Un appui simple permet une structure suivante:

{ LAM } // Ici mettez les noms de variables à définir LAM A LAM B ...

// La dernière variable prend pour valeur l'objet 1 de la pile...

BIND // Début de la section locale

ABND // Fin d'une section locale

Un shift droit permet l'usage des variables locales sans nom et un shift gauche de variable locales dont le nom est perdu à la compilation.

Dans le dernier cas la structure est:

{{ }} // Mettez vos variable Var1 Var2 ...

ABND // Fin d'une section locale

3) La troisième option, permet les boucles. Toute condition provoque un des deux external TRUE ou FALSE. Un de ces externale doit être sur la pile avant un test (saut conditionnelle, boucle...)

La boucle que j'utilise tout le temps est:

BEGIN // Début de la boucle

... // code

UNTIL // fin de la boucle

Si l'external FALSE est sur la pile avant UNTIL recommence à BEGIN et si c'est TRUE la boucle se termine.

Une autre boucle existe avec shift-gauche, soit en remplaçant UNTIL par AGAIN. C'est la même sauf que l'external doit être TRUE au lieu de FALSE.

4) La quatrième option permet la mise en place de trappe en cas d'erreur - si je ne me trompe pas; je m'en suis jamais servi.

5-6) Sert pour l'assembleur saturn.

La structure:

CODE

... // Assembleur

ENDCODE

Permet d'inclure de l'assembleur au milieu d'un programme sysRPL.

**3.3 Les boucles conditionnelles.**

Il reste à expliqué comment avoir des boucle type IF THEN END et dérivé.

C'est très simple. La commande IT (**I**f **T**hen) permet de n'exécuté la commende suivant que si TRUE est au niveau 1 de la pile. Pour exécuté plusieurs commende on crée un bloc qui commence par :: et finit par; .

On a alors une structure type:

Condition

IT :: *Votre code* ;

Pour une structure type IF THEN ELSE, on a la commande ITE et le même principe, soit une structure:

*Condition*

ITE :: *Code si vrai* ;

:: *Code si faux* ;

**3.4 Introduction de code userRPL**

Il est possible d'utiliser les commandes userRPL, en les précédant d'un x minuscule.

Par exemple, *x+* permet d'obtenir la commande + en userRPL.

**Conclusion**

J'espère qu'à la fin de ceux guide vous saurez asses pour programmer facilement en sysRPL, langage plus rapide que l'userRPL, tout en étant plus facile que l'assembleur.