Algorithmique
et traduction pour calculatrices
et MuPAD

1 Pour commencer

1.1 Comment éditer et sauver un programme

1.1.1 Traduction MuPAD

Il est commode d'utiliser `emacs` comme interface, et d'avoir deux *Buffers* ouverts : l'un deux sera le texte du programme, l'autre sera une session `*MuPAD*` qui permettra de tester que le programme est correct (il faudra parfois ouvrir un troisième Buffer pour une session de mise au point lorsqu'il se produit une erreur un peu récalcitrante).
Lancez `emacs` puis tapez sur la touche `Echap` puis, tapez `x mupad` < **Return**>
puis `Start MuPAD` du menu `MuPAD` ce qui lance une session `*MuPAD*` comme d'habitude. Dans le menu `Files` sélectionnez `Open File` et donnez, par exemple, comme nom de fichier `prog1.mu`. Dans le menu `Buffers` vous devez avoir les buffers suivants `*MuPAD*` et `prog1.mu`, il suffit de cliquer sur le nom de buffer pour passer de la session `*MuPAD*` à l'édition du programme.

Nous allons écrire une fonction appelée `ma_procedure` qui calculera la moyenne de deux nombres.

Sélectionnez le buffer `prog1.mu`, puis dans le menu `MuPAD` sélectionnez `Shapes` puis `Procedure`. On vous demande dans l'ordre d'entrer le nom de la procédure, par exemple `ma_procedure`, puis les arguments (ou paramètres) qu'il faudra lui passer, par exemple `x1,x2`, puis les options (tapez sur < **Return**> pour ne pas donner d'options). Votre texte doit ressembler à :

```
ma_procedure:=
proc (x1,x2)
begin

end_proc: /* End of ma_procedure */
```

Il ne reste plus qu'à entrer entre `begin` et `end_proc` :
`return (x1+x2)/2;`
et à sauvegarder le fichier (`Save Buffer` du menu `Files`).

Sélectionnez maintenant le buffer `*MuPAD*` et tapez :
`read("prog1.mu");`
ce qui charge le fichier texte qui contient la définition de `ma_procedure`. Vous pouvez maintenant calculer la moyenne des deux entiers 5 et 13 en tapant :
`ma_procedure(5,13);`
qui doit vous renvoyer `9`.

1.1.2 Traduction Casio

Appuyer sur la touche `MENU`, puis ouvrir le menu `PRGM` (en mettant en surbrillance `PRGM` avec les flèches, puis en appuyant sur la touche `EXE`).
Cela vous donne la liste des programmes (écran `Program List`) et un bandeau contenant : `EXE EDIT NEW DEL DEL.A SRC REN`
Avec `F3` vous sélectionnez `NEW` pour éditer un nouveau programme.
On vous demande `Program Name` : il suffit de taper le nom de votre programme (vous êtes en mode Alpha!).
Les touches `SHIFT VARS (PRGM)` font apparaître un bandeau contenant les différentes instructions et la touche `OPTN` fait apparaître un bandeau contenant les différentes fonctions mathématiques.
La touche `EXIT` permet de sortir d'une application.
Pour sauver votre programme, il suffit de taper `2nd EXIT (QUIT)` et on revient à l'écran `Program List`.

1.1.3 Traduction HP38 HP40G

Pour avoir accès au catalogue de programmes, on appuie sur les touches `shift 0 (PROGRAM)`.
Il apparait alors un écran contenant la liste des programmes disponibles et un bandeau (`EDIT NEW SEND RECV RUN`).
Pour taper un nouveau programme, on appuie sur `F2 (NEW)`.
On vous demande le nom du programme : attention! vous n'êtes pas en mode `Alpha` appuyer sur `F4 (A..Z)` pour y être!!!.
Tapez son nom puis `F6 (OK)`.
Vous entrez votre programme et votre travail est automatiquement sauvegardé.

1.1.4 Traduction HP48 HP49G mode RPN

La `HP49G` peut travailler selon deux modes : le mode RPN qui est la notation polonaise inverse et où les différents résultats sont mis sur une pile, ou le mode algébrique qui est la notation habituelle (cf 1.1.5).
Un programme s'écrit dans la ligne de commande entre les délimiteurs ≪ ≫
Pour le sauver, il suffit de faire suivre le dernier ≫ par :

`'NOMDUPROGRAMME' STO▷`

1.1.5 Traduction HP49G mode Algébrique

Un programme s'écrit dans la ligne de commande entre les délimiteurs ≪ ≫
Pour le sauver, il suffit de faire suivre le dernier ≫ par :

`STO ▷ NOMDUPROGRAMME`

1.1.6 Traduction TI 80

On appuie sur la touche `PRGM` et on met en surbrillance `NEW` avec la flèche →, puis `ENTER`.
On obtient :
`1 :CREATE NEW`
Il suffit de taper `ENTER`.
On obtient :`NAME=`
Il suffit de taper le nom du programme (vous êtes en mode `ALPHA`!), puis `ENTER`.
Vous êtes dans l'éditeur et vous pouvez taper votre programme (cf 1.1.7).

1.1.7 Traduction TI 83+

On appuie sur la touche `PRGM` et on met en surbrillance `NEW` avec la flèche →, puis `ENTER`.
On obtient :
`PROGRAM Name=`
Il suffit de taper le nom du programme (vous êtes en mode `ALPHA`!), puis `ENTER`.
Vous êtes dans l'éditeur et vous pouvez taper votre programme :
- les `:` sont mis automatiquement en début de ligne.
- les différentes instructions sont disponibles en appuyant sur `PRGM` : en effet, depuis l'éditeur le menu de `PRGM` n'est pas le même (`CTL` pour les différentes instructions, `I/O` pour les instructions d'entrées et de sorties, et enfin `EXEC` pour afficher la liste des noms des programmes ce qui permet d'écrire `prgmNOMDUPROG` dans une ligne de programme pour appeler un sous programme).
- les fonctions mathématiques sont accessibles en appuyant sur `MATH` et les fonctions booléennes en appuyant sur `2nd MATH (TEST)`.
Pour sauver votre programme, il suffit de taper `2nd MODE (QUIT)` et vous revenez à l'écran `HOME`.

1.1.8 Traduction TI89, 92

Pour avoir accès à l'éditeur, on appuie sur la touche `APPS` puis `7 (Program Editor)` puis `3 (New)` et `ENTER`.
On vous demande, dans `Type`, si on veut écrire une fonction (qui renvoie une valeur grâce à `Return`) ou un programme (appuyer `->` puis faites votre choix `1` ou `2`) puis `ENTER` pour valider votre choix.
On écrit ensuite le nom du programme dans la case `Variable` puis `ENTER` pour valider le nom, puis `ENTER` pour

entrer dans l'éditeur.
Vous êtes prêt à taper votre programme.
Vous pouvez ajouter éventuellement les noms des paramètres dans la première ligne (en déplaçant le curseur avec les flèches).
Les touches `F1 F2 F3 F4` vous facilitent l'édition.
En appuyant sur ♦ `Q` (`HOME`) vous sauvez votre programme et vous revenez à l'écran `HOME`.

1.1.9 Traduction SHARP EL9600

On appuie sur la touche `2nd MATRIX (PRGM)` et on met en surbrillance `NEW` avec la flèche ↓ et `ENTER` (ou bien avec le stylet il suffit de toucher `NEW` deux fois).
On obtient :
`TITLE ?`
Il suffit de taper le nom du programme (vous êtes en mode `ALPHA` !), puis `ENTER`.
Vous êtes dans l'éditeur et vous pouvez taper votre programme :
- les différentes instructions sont disponibles en appuyant sur `2nd MATRIX (PRGM)` : en effet, depuis l'éditeur le menu de `2nd MATRIX (PRGM)` n'est pas le même (`PRGM` pour les instructions d'entrées et de sorties, `BRNCH` pour les différentes instructions de programmation etc...).
- les fonctions mathématiques sont accessibles en appuyant sur `MATH` (`CALC` et `NUM` donnent les fonctions de calculs et d'arrondis, `INEQ` donne les symboles d'inégalités et `LOGIC` donne les fonctions booléennes).
Pour sauver votre programme, il suffit de taper `2nd CL (QUIT)` et on revient à l'écran `HOME`.

1.2 Comment corriger un programme

1.2.1 Traduction MuPAD

Si la syntaxe est mauvaise, la machine vous indique un message d'erreurs.
Faites apparaître le texte de votre programme : ouvrir le menu `Buffers` et sélectionner le nom de votre programme (`prog1.mu`).
Vous pouvez corriger votre erreur, puis sauvegarder votre correction dans le fichier `prog1.mu` (`Save Buffer` du menu `Files`). Sélectionnez maintenant le buffer `*MuPAD*` et tapez :
`read("prog1.mu");`
ce qui charge le fichier texte qui contient la définition de `ma_procedure`.
Si vous ne voyez pas quelle peut être la source d'une erreur, vous pouvez utiliser le débuggueur `mdx`. Le programme `mdx` permet de stopper l'exécution d'un programme `MuPAD` à une ligne donnée et continuer l'exécution ligne après ligne avec la possibilité d'examiner le contenu des variables.

Pour lancer `mdx`, choisissez `Debug` du menu `MuPAD` ou tapez dans la fenêtre `emacs` la touche `Echap` puis `x mdx` puis < **Return**>.
Vous devez voir apparaitre :
`Run mdx (like this): mupad prog1.mu`
Tapez `Entree` (après avoir changé le nom du fichier si nécessaire).

Vous devez voir apparaitre un buffer qui ressemble fortement au buffer `*MuPAD*`. Vous pouvez maintenant exécuter en mode pas à pas n'importe quelle procédure de votre fichier en tapant par exemple :
`debug(ma_procedure(5,13));`
La fenêtre `emacs` doit maintenant être divisée en deux parties, avec le texte source d'une part et le buffer `*gud-prog1.mu*` d'autre part. Les commandes de mise au point doivent être écrites dans le buffer `*gud-prog1.mu*`.
Les principales sont :
- `p` affiche la valeur d'une variable
- `n` exécute la ligne courante du source
- `s` comme `n` mais en exécutant les appels de procédure de la ligne courante en mode pas-à-pas
- `c` continue l'exécution jusqu'au prochain point d'arrêt. Il est possible de mettre un point d'arrêt en mettant dans le texte source, le curseur là où on veut arrêter l'exécution du programme puis en tapant `Ctrl-X Ctrl-A Ctrl-B` dans la fenêtre source (ce qui exécute la commande `S` dans la fenêtre du debuggeur).

Pour en savoir plus sur la programmation, vous pouvez par exemple consulter avec Netscape les notes de B. Ycart et P. Zimmermann :
`http://www.math-info.univ-paris5.fr/Enseignements/demarre_mupad/`

1.2.2 Traduction Casio

Si la syntaxe est mauvaise, la machine vous indique : `Go ERROR` si vous voulez aller là où se trouve l'erreur appuyer sur l'une des flèches ($\leftarrow, \rightarrow$).
Vous êtes alors dans l'éditeur prêt à corriger votre erreur.
Attention!!!
- vous n'êtes pas en mode insertion automatiquement (appuyer sur `2nd DEL (INS)` pour être en mode insertion jusqu'au prochain déplacement du curseur).

1.2.3 Traduction HP38 HP40G

Si la syntaxe est mauvaise, la machine vous dit :
`Invalid Syntax Edit program ?` Vous répondez `F6 (YES)`.
La machine vous met automatiquement le curseur là où le compilateur a détecté l'erreur. Il suffit donc de corriger!!!

1.2.4 Traduction HP48 HP49G mode RPN

Si la syntaxe est mauvaise, la machine vous met automatiquement le curseur là où le compilateur a détecté l'erreur. Il suffit donc de corriger!!!
Si l'erreur est détectée au cours de l'exécution du programme il faut taper :
`'NOMDUPROGRAMME' VISIT` qui édite votre programme.
On corrige, puis `ENTER` sauve votre programme corrigé.
Si vous ne voyez pas quelle peut être la source d'une erreur, vous pouvez utiliser le débuggueur :
- sur la `HP49G` taper `shift-bleu CAT (PRG)` et séléctionner le menu `14 RUN & DEBUG` puis `ENTER`.
- sur la `HP48G` taper `PRG` puis `NXT` puis ouvrir le répertoire `RUN` du bandeau.

Le bandeau `DEBUG SST SST` ↓ `NEXT HALT2 KILL` s'affiche.
`DEBUG` lance le programme et exécute la première instruction puis, `NEXT` affiche l'étape suivante sans l'exécuter et `SST` l'exécute (`NEXT SST` exécute le programme au pas à pas).
Si l'on veut faire l'exécution du programme en pas à pas à partir du milieu du programme il faut mettre un point d'arrêt à cet endroit en insérant la commande `HALT` dans le programme. On lance alors le programme qui s'arrête au niveau du `HALT`, on fait l'exécution en pas à pas comme précédemment avec `NEXT SST` etc..
On utilise `KILL` pour arrêter le debugger quand on a compris où etait l'erreur et `Shift-rouge ON (CONT)` pour continuer l'exécution du programme.

1.2.5 Traduction HP49G mode Algébrique

Si la syntaxe est mauvaise, la machine vous met automatiquement le curseur là où le compilateur a détecté l'erreur. Il suffit donc de corriger!!!
Si l'erreur est détectée au cours de l'exécution du programme il faut taper :
`VISIT('NOMDUPROGRAMME')` qui édite votre programme.
On corrige, puis `ENTER` sauve votre programme corrigé.

1.2.6 Traduction TI 80 TI 83+

Si la syntaxe est mauvaise, la machine vous demande si vous voulez aller là où se trouve l'erreur (`1 :GOTO`) ou si vous voulez arrêter (`2 :QUIT`).
Vous répondez : 1
Vous êtes alors dans l'éditeur prêt à corriger votre erreur.
Puis `2nd MODE (QUIT)` vous fait revenir à l'écran `HOME` en sauvant votre correction.

1.2.7 Traduction TI89 92

Si la syntaxe est mauvaise, la machine vous demande si vous voulez aller là où se trouve l'erreur (`ENTER=GOTO`) ou si vous voulez arrêter (`ESC=CANCEL`).
Vous répondez : `ENTER`.
Vous êtes alors dans l'éditeur prêt à corriger votre erreur.
Puis ♦ `Q` (`HOME`) vous fait revenir à l'écran `HOME` en sauvant votre correction.

Si l'erreur est détectée au cours de l'exécution du programme il faut taper :
`APPS` puis `7 (Program Editor)` puis `1 (Current)` et `ENTER` qui édite votre programme.

1.2.8 Traduction SHARP EL9600

Si la syntaxe est mauvaise, la machine vous demande si vous voulez aller là où se trouve l'erreur à l'aide des flèches ($\leftarrow, \rightarrow$ `:Goto error`) ou si vous voulez arrêter (`CL :Quit`),.
Vous répondez en appuyant sur une flèche : vous êtes alors dans l'éditeur prêt à corriger votre erreur.
Attention!!!
- vous n'êtes pas en mode insertion automatiquement (appuyer sur `2nd DEL (INS)` pour être en mode insertion jusqu'au prochain `2nd DEL (INS)`).
- il faut valider la ligne corrigée en changeant de ligne en appuyant par exemple sur $\downarrow$.

Puis `2nd CL (QUIT)` vous fait revenir à l'écran `HOME`.

1.3 Comment exécuter un programme

1.3.1 Traduction MuPAD

Vous sélectionnez maintenant le buffer `*MuPAD*` et tapez :
`read("prog1.mu");`
ce qui charge le fichier texte qui contient la définition de `ma_procedure` (qui calcule par exemple la moyenne de deux nombres cf exemple `??`). Vous pouvez maintenant calculer la moyenne des deux entiers 5 et 13 en tapant :
`ma_procedure(5,13);`
qui doit vous renvoyer `9`.

1.3.2 Traduction Casio

Dans l'écran `Program List` (au départ de l'écran `HOME` touche `MENU PRGM EXE` ou après avoir tapé le programme `2nd EXIT (QUIT)`), il suffit de mettre en surbrillance le nom du programme à exécuter et de taper sur `F1` pour `EXE` du bandeau (dernière ligne de l'écran où se trouve un menu).

1.3.3 Traduction HP38 HP40G

Pour exécuter un programme, on ouvre le catalogue de programmes en appuyant sur les touches `shift 0 (PROGRAM)`.
Il apparait alors un écran contenant la liste des programmes disponibles et un bandeau (`EDIT NEW SEND RECV RUN`).
On met le nom du programme à exécuter en surbrillance et on appuie sur `F6 (RUN)`.

1.3.4 Traduction HP48 HP49G mode RPN

Si le programme n'a pas de paramètres, il suffit de taper son nom dans la ligne de commande ou d'utiliser le menu `VAR`.
S'il y a des paramètres, on met les valeurs des paramètres sur la pile puis on tape le nom du programme.
Exemple :
`45 75 PGCD`

1.3.5 Traduction HP49G mode Algébrique

Si le programme n'a pas de paramètres, il suffit de taper son nom dans la ligne de commande ou d'utiliser le menu `VAR`.
S'il y a des paramètres, on fait suivre le nom du programme de parenthèses dans lesquelles on met les valeurs des paramètres séparées par une virgule.
Exemple :
`PGCD(45,75)`

1.3.6 Traduction TI 80

On appuie sur la touche `PRGM`, on laisse en surbrillance `EXEC`, et on tape le numéro du programme à exécuter.

1.3.7 Traduction TI 83+

On appuie sur la touche PRGM, on laisse en surbrillance EXEC, et on tape le numéro du programme à exécuter.

1.3.8 Traduction TI89 92

Si le programme n'a pas de paramètres, il suffit de taper son nom dans la ligne de commande.
S'il y a des paramètres, on fait suivre le nom du programme de parenthèses dans lesquelles on met les valeurs des paramètres séparées par une virgule.
Exemple :
`PGCD(45,75)`

1.3.9 Traduction SHARP EL9600

On appuie sur la touche 2nd MATRIX (PRGM), on met en surbrillance EXEC, et le programme à exécuter avec le stylet ou on utilise les flèches ou on tape le numéro (à deux chiffres) du programme à exécuter.

1.4 Comment améliorer puis sauver sous un autre nom un programme

1.4.1 Traduction MuPAD

Si vous voulez avoir à la fois l'ancien et le nouveau programme il faut :
faire apparaître le texte de votre programme en ouvrant le menu Buffers et sélectionner le nom de votre programme (prog1.mu).
Vous pouvez corriger améliorer votre programme (pensez à changer le nom de la procédure (par exemple ma_procedure2), puis sauvegarder votre correction dans le fichier prog2.mu (par exemple) avec Save Buffer As... du menu Files : il faut alors donner le nom du nouveau fichier (ici prog2.mu).
Sélectionnez maintenant le buffer *MuPAD* et tapez :
`read("prog2.mu");`
ce qui charge le fichier texte qui contient la définition de ma_procedure2.

1.4.2 Traduction Casio

Dans l'écran Program List (MENU PRGM EXE), il suffit de mettre en surbrillance le nom du programme à changer et de taper sur F6 puis F2 pour REN (RENname) du bandeau.
Mais cela ne duplique pas le programme!!! on doit donc tout retaper pour améliorer...

1.4.3 Traduction HP38 HP40G

Pour modifier un programme (sans vouloir sauver l'ancien) on ouvre le catalogue de programmes, en appuyant sur les touches shift 0 (PROGRAM). Il apparait alors un écran contenant la liste des programmes disponibles et un bandeau (EDIT NEW SEND RECV RUN).
On met le nom du programme à modifier en surbrillance et on appuie sur F1 (EDIT).
Si vous voulez avoir à la fois l'ancien et le nouveau programme il faut :
-ouvrir le catalogue de programmes (shift 0 (PROGRAM))
-appuyer sur F2 (NEW) et taper le nom du programme modifié puis F6 (OK).
L'éditeur s'ouvre, on appuie alors sur VAR puis A..Z 5 (P) pour mettre Program en surbrillance.
Avec les flèches mettre le nom du programme à modifier en surbrillance et appuyer sur F4 (VALUE) (pour cocher VALUE du bandeau) puis F6 (OK).
Cela recopie le texte du programme dans l'éditeur.

1.4.4 Traduction HP48 HP49G mode RPN

On tape :
'NOMDUPROGRAMME' RCL puis edit du bandeau.
On fait les améliorations et on fait suivre le dernier ≫ par :

'NOUVEAUNOM' STO▷

1.4.5 Traduction HP49G mode Algébrique

On tape :
`RCL('NOMDUPROGRAMME')` puis `edit` du bandeau.
On fait les améliorations et on fait suivre le dernier ≫ par :

`STO ▷ NOUVEAUNOM`

1.4.6 Traduction TI 80

Je n'ai pas trouvé comment faire...

1.4.7 Traduction TI 83+

Vous commencez comme si vous alliez écrire un nouveau programme (`PRGM NEW` puis le nom du nouveau programme).
Vous êtes alors dans l'éditeur, vous avez la possibilité d'éditer n'importe quel programme avec :
`2nd STO (RCL)` puis `PRGM`, puis, mettre en surbrillance `EXEC` puis, le numéro du programme à rappeler.
En bas de l'éditeur il s'écrit : `Rcl prgmPGCD` (par exemple).
`ENTER` ramène alors toutes les lignes du programme `PGCD` dans l'éditeur.

1.4.8 Traduction TI89 92

Vous ouvrez l'éditeur avec le programme que vous voulez améliorer puis sauver sous un autre nom :
`APPS` puis `7 (Program Editor)` puis `2 (Open)` puis mettre le nom du programme à modifier dans `Variable` (appuyer sur `->` pour avoir accès à tous les noms de vos programmes, sélectionner celui à modifier avec les flèches et `ENTER`), puis `ENTER`.
Ensuite, il suffit de taper sur ♦ `S` pour sauver cet éditeur sous un autre nom : `SAVE COPY AS` (on met le nouveau nom dans `Variable`).
Vous pouvez faire vos modifications puis ♦ `Q` (`HOME`) vous sauve votre programme modifié sous le nouveau nom, et vous revenez à l'écran `HOME`.

1.4.9 Traduction SHARP EL9600

On peut copier une ligne grâce au sous menu `COPY` du menu `2nd MATRIX (PRGM)`
On ne peut pas dupliquer un programme...on doit donc retaper pour améliorer...

2 Les différentes instructions

2.1 Les commentaires

Il faut prendre l'habitude de commenter les programmes. En algorithmique un commentaire commence par `//` et se termine par un passage à la ligne.

2.1.1 Traduction MuPAD

un commentaire est entouré de deux `#` ou
commence par `/*` et se termine par `*/` ou
commence par `//` et se termine par un passage à la ligne.

```
# ceci est un commentaire #
/* ceci est un commentaire */ //ceci est un commentaire
```

2.1.2 Traduction Casio

Il n'y en a pas!!!

2.1.3 Traduction HP38 HP40G 48 49

Le commentaire commence par :
`@` et se termine par un passage à la ligne.
Pour la `HP49G`, un commentaire commence par `@` et se termine par un passage à la ligne ou est entouré de deux `@`.
Le caractère `@` est obtenu en tapant `shift-rouge ENTER`
Attention!!! le compilateur efface les commentaires... donc pour garder vos commentaires, il faut écrire votre programme sous la forme d'un texte qu'il faut ensuite compiler avec `STR` → ce qui complique un peu...

2.1.4 Traduction TI80 83+

Il n'y en a pas!!!

2.1.5 Traduction TI89 92

Le commentaire commence par
© (F2 9) et se termine par un passage à la ligne.

2.1.6 Traduction SHARP EL9600

Les commentaires sont précédés de `Rem` (que l'on trouve dans le sous menu `PRGM` du menu `2nd MATRIX (PRGM)` lorsqu'on est en train d'éditer un programme).

2.2 Les variables

2.2.1 Leurs noms

Ce sont les endroits où l'on peut stocker des valeurs, des nombres, des expressions. Les noms des variables sont soit prédéfinis soit plus libres.
Par exemple chez `CASIO, HP38 HP40G, TI80, TI83+ et SHARP-EL9600` on n'a droit qu'aux 26 lettres de l'alphabet pour stocker des nombres réels.
Mais avec `MuPAD`, la `TI 89`, la `TI92`, la `HP48` ou la `HP49G` on peut utiliser des noms parfois limités à 8 caractères.

2.2.2 Notion de variables locales

Cette notion n'existe pas pour les calculatrices `CASIO, HP38 HP40G, TI80, TI83+ et SHARP-EL9600`.
Pour `MuPAD` et les `TI92 TI89` il faut définir les variables locales en début de programme en écrivant :
`local a,b`
La `HP48` ou `HP49G` peut utiliser des variables locales.
Les variables locales sont déclarées et initialisées (initialisation obligatoire!) grâce à → `(shift-rouge 0)`
Pour la `HP48` ou `HP49G mode RPN` il faut définir et initialiser les variables locales en écrivant :

$$\ll 1\ 2 \rightarrow \mathtt{A\ B} \ll \text{corps du programme} \gg\gg$$

pour initialiser `A` à 1 et `B` à 2.
En mode `RPN` on peut définir et initialiser plusieurs variables locales à la fois (cf exemple ci-dessus).
Pour la `HP49G mode Algébrique` chaque déclaration doit être suivie par un sous programme (délimiteurs ≪≫) en écrivant :

$$\ll 1 \rightarrow \mathtt{A} \ll 2 \rightarrow \mathtt{B} \ll \text{corps du programme} \gg\gg\gg$$

La flèche doit être entourée d'espaces, ces espaces sont mis automatiquement quand on n'est pas en mode `Alpha`.
Exemple :

$$\ll 3.14 \rightarrow \mathtt{PI} \ll 2 * \mathtt{PI} * \mathtt{R} \gg\gg \mathtt{STO}\triangleright\ \mathtt{PER}$$

Dans cet exemple, on a écrit le programme `PER`.
`PI` est une variable locale qui est déclarée et affectée par `3.14` → `PI`. Cette variable est locale pour le programme qui suit sa déclaration (ici ≪ 2 ∗ PI ∗ R ≫).
Par contre, `R` est une variable globale (qui doit exister avant l'exécution du programme `PER`). Si, au cours d'un programme, on veut stocker une valeur dans une variable (locale ou globale) il faut bien sûr utiliser `STO▷`.

2.2.3 Notion de paramètres

Quand on écrit une fonction il est possible d'utiliser des paramètres.
Par exemple si A et B sont les paramètres de la fonction PGCD on écrit :
avec MuPAD
```
PGCD :=proc(A,B)
begin
....
end_proc
```
avec les calculatrices l'en tête sera de la forme PGCD(A,B)
Ces paramètres se comportent comme des variables locales, la seule différence est qu'ils sont initialisés lors de l'appel de la fonction. L'exécution se fait en demandant par exemple : PGCD(15,75)
Il n'est pas possible d'écrire des fonctions sur toutes les calculatrices.
Pour les TI 89 92 on met le nom des paramètres dans le nom de la fonction par exemple :
:addition(a,b)
Par exemple, pour la HP48 ou HP49G mode RPN si on veut que R soit le paramètre de la fonction PER on écrit :

$$\ll \rightarrow \text{R} \ll 3.14 \rightarrow \text{PI} \ll 2\ \text{PI} * \text{R} * \gg\gg\gg' \text{PER}' \text{ STO}\triangleright$$

L'exécution se fait en demandant par exemple : 5 PER.
Pour la HP49G mode Algébrique si on veut que R soit le paramètre de la fonction PER on écrit :

$$\ll \rightarrow \text{R} \ll 3.14 \rightarrow \text{PI} \ll 2 * \text{PI} * \text{R} \gg\gg\gg \text{STO}\triangleright \text{ PER}$$

L'exécution se fait en demandant par exemple : PER(5).
La syntaxe des HP48 et HP49G en mode RPN et en Algébrique est :

$$\ll \rightarrow \text{A B} \ll \ldots \gg\gg$$

pour écrire une fonction à deux paramètres. **Remarque** Le langage de MuPAD et des HP48/49 est fonctionnel puisque on peut passer des programmes ou des fonctions en paramètre.

2.3 Les Entrées

Pour que l'utilisateur puisse entrer une valeur dans la variable A au cours de l'exécution d'un programme, on écrira, en algorithmique :
saisir A
Et pour entrer des valeurs dans A et B on écrira :
saisir A,B

2.3.1 Traduction MuPAD

```
input("a=",a)
input("a=",a,"b=",b )
```
mais la plupart des entrées se font par passage de paramètres.

2.3.2 Traduction Casio

```
"A=" ?->A :
```

2.3.3 Traduction HP38 HP40G

```
INPUT A;"TITRE";"A=";;0 :
```

2.3.4 Traduction HP48 HP49G mode RPN

```
"A" "" INPUT STR-> EVAL 'A' STO
```
ou pour la HP49G mode RPN
```
'A' PROMPTSTO
```

2.3.5 Traduction HP49G mode Algébrique

Pour entrer une variable A, on écrit :
`...PROMPTSTO('A')`

2.3.6 Traduction TI 80, 82, 83, 83+, 89, 92

```
 :Prompt A
:Prompt A,B
```
ou encore :
`:Input "A=",A`

2.3.7 Traduction SHARP EL9600

`Input A`
La machine met alors automatiquement `A=?` (ou `A=?` puis `B=?`).

2.4 Les Sorties

En algorithmique on écrit :
`Afficher "A=",A`

2.4.1 Traduction MuPAD

```
print("A=",A)
```
Il fait savoir que lorsqu'une procèdure est appelée, la suite d'instructions entre `begin` et `end_proc` est ecécutée, et le résultat est égal au résultat de la dernière évaluation.

2.4.2 Traduction Casio

`"A=" :A` △
`ClrText` efface l'écran.
△ arrête l'affichage et sert aussi de fin d'instruction.

2.4.3 Traduction HP38 HP40G

`DISP 3;"A="A` : 3 représente le numéro de la ligne où `A` sera affiché
ou
`MSGBOX "A="A :`
`ERASE` : efface l'écran.
`FREEZE` : gèle l'affichage.

2.4.4 Traduction HP48 ou HP49G mode RPN

En général on affiche simplement les résultats sur la pile pour une réutilisation éventuelle, on écrira simplement :
`A B`
On peut aussi afficher le résultat tagué, on écrira alors :
`A "A=" ->TAG`
`HALT` arrête le programme et `Shift-rouge ON (CONT)` le continue.

2.4.5 Traduction HP49G mode Algébrique

Seul le dernier résultat s'inscrit dans l'historique.
Pour des résultats intermédiaires on tape :

$$\texttt{DISP}(\text{"A} = \text{"} + \texttt{A}, 3)$$

3 représente le numéro de la ligne
ou

$$\texttt{MSGBOX}(\text{"}\texttt{A} = \text{"} + \rightarrow \texttt{STR}(\texttt{A}))$$

(ici le + effectue la concaténation de deux chaines de caractères)
`CLLCD()` efface l'écran.
`FREEZE(7)` gèle l'affichage et permet de visualiser les 7 lignes de l'affichage.

2.4.6 Traduction TI 80 82 83 83+ 89 92

`:Disp "A=",A`
`:ClrIO` efface l'écran.
`:ClrHome` efface l'écran pour la `TI 83+`.
`:Pause` arrête le programme (on appuie sur `ENTER` pour reprendre l'exécution).

2.4.7 Traduction SHARP EL9600

`Print A` affiche la valeur de `A`
ou `Print "A` affiche `A`, puis
`Print A` affiche la valeur de `A`

2.5 La séquence d'instructions ou action

Une action est une séquence d'une ou plusieurs instructions.
En langage algorithmique, on utilisera l'espace ou le passage à la ligne pour terminer une instruction.

2.5.1 Traduction MuPAD

Le `;` est un séparateur d' instructions.

2.5.2 Traduction Casio

A la fin d'une instruction, il faut mettre un séparateur qui peut être :
`le retour à la ligne` ou `:` ou $\triangle$

2.5.3 Traduction HP38 HP40G

`:` indique la fin d'une instruction.

2.5.4 Traduction HP48 ou HP49G mode RPN

Comme en algorithmique, le séparateur est soit `l'espace` soit `le retour à la ligne`.

2.5.5 Traduction HP49G mode Algébrique

Pour la `HP49G`, le `;` est un séparateur d' instructions.
Le `;` s'obtient en tapant en même temps sur `shift-rouge SPC`.

2.5.6 Traduction TI 83+ 89 92

`:` indique la fin d'une instruction. Il faut noter qu'à chaque passage à la ligne le `:` est mis automatiquement.

2.5.7 Traduction SHARP EL9600

C'est `ENTER` qui est utilisé comme séparateur d'instructions.
Chaque instruction doit être écrite sur une seule ligne.

2.6 L'instruction d'affectation

L'affectation est utilisée pour stocker une valeur ou une expression dans une variable.
En algorithmique on écrira par exemple : `2*A->B`
pour stocker `2*A` dans `B`
Les calculatrices `CASIO, HP38 HP40G, TI80, TI83+, TI89, TI92 et SHARP-EL9600` utilisent cette notation.
Selon les cas, la flèche est obtenue à l'aide de la touche `STO` ou de la touche →
Avec la `HP48` ou `HP49G mode RPN`, il faut utiliser la notation postfixée et la commande `STO` :
`2 A * 'B' STO`
Pour la `HP49G mode Algébrique`, on utilise la touche `STO` qui se traduit à l'écran de la calculatrice par : ▷ (que l'on notera : `STO▷`)
Pour la `SHARP EL9600`, on utilise la touche `STO` qui se traduit à l'écran de la calculatrice par : ⇒
ATTENTION
Avec `MuPAD` on écrit :
`b :=2*a` pour stocker `2*a` dans `b`.

2.7 Les instructions conditionnelles

`Si` *`condition`* `alors` *`action`* `fsi`
`Si` *`condition`* `alors` *`action1`* `sinon` *`action2`* `fsi`
Exemple :
`Si A = 10 ou A < B alors B-A->B sinon A-B->A fsi`

2.7.1 Traduction MuPAD

`if` *`condition`* `then` *`action`* `end_if`
`if` *`condition`* `then` *`action1`* `else` *`action2`* `end_if`
Exemple :
`if a = 10 or A < B then b :=b-a else a :=a-b end_if`
Lorsque il y a plusiuers `if else` à la suite on écrit `elif` au lieu de `else if`.

2.7.2 Traduction Casio

`If` *`condition`* `:Then` *`action`* `: IfEnd :`
`If` *`condition`* `:Then` *`action1`* `: Else :` *`action2`* `: IfEnd :`
Exemple :
`If A = 10 Or A < B :Then B-A->B : Else : A-B->A : IfEnd :`

2.7.3 Traduction HP38 HP40G

`IF` *`condition`* `THEN` *`action`* `: END :`
`IF` *`condition`* `THEN` *`action1`* `: ELSE` *`action2`* `: END :`
Exemple (Attention au == pour traduire la condition d'égalité) :
`IF A == 10 OR A < B THEN B-A->B : ELSE A-B->A : END :`

2.7.4 Traduction HP48 ou HP49G mode RPN

`IF` *`condition`* `THEN` *`action`* `END`
`IF` *`condition`* `THEN` *`action1`* `ELSE` *`action2`* `END`
Attention on utilise la notation postfixée et == pour traduire la condition d'égalité.
On écrit pour traduire l'exemple :
`IF A 10 == A B < OR THEN B A - 'B' STO ELSE A B - 'A' STO END`
on peut aussi écrire :
`IF '(A==10) OR (A < B)' THEN ...`

2.7.5 Traduction HP49G mode Algébrique

`IF` *`condition`* `THEN` *`action`* `END`
`IF` *`condition`* `THEN` *`action1`* `ELSE` *`action2`* `END`

Exemple (Attention au == pour traduire la condition d'égalité) :
`IF A == 10 OR A < B THEN B-A STO▷ B ELSE A-B STO▷ A END`

2.7.6 Traduction TI80

` :If condition :Then : action : End`
`:If condition :Then : action1 : Else : action2 : End`
Exemple :
Attention or n'existepas il faut donc écrire (cf p 16) : `:IF A = 10 :THEN : B-A->B :ELSE :IF A<B :THEN : B-A->B :ELSE : A-B->A :END :END`

2.7.7 Traduction TI82 83+

` :If condition :Then : action : End`
`:If condition :Then : action1 : Else : action2 : End`
Exemple :
`:If A = 10 or A < B : Then : B-A->B : Else : A-B->A : End`

2.7.8 Traduction TI89 92

` :If condition Then : action : EndIf`
`:If condition Then : action1 : Else : action2 : EndIf`
Exemple :
`:If A = 10 or A < B Then : B-A->B : Else : A-B->A : EndIf`

2.7.9 Traduction SHARP EL9600

`If non condition Goto FSI`
action
`Label FSI`

`If non condition Goto SINON`
action1
`Goto FSI`
`Label SINON`
action2
`Label FSI`

2.8 Les instructions "Pour"

`Pour I de A à B faire action fpour`
`Pour I de A à B (pas P) faire action fpour`

2.8.1 Traduction MuPAD

`for i from a to b do action end_for`
`for i from b downto a do action end_for`
`for i from a to b step p do action end_for`
Vous pouvez aussi ouvrir le menu `MuPAD` sous menu `Shapes` et séléctionner `for`.

2.8.2 Traduction Casio

`For A->I To B : action : Next`
`For A->I To B Step P : action : Next`

2.8.3 Traduction HP38 HP40G

`FOR I = A TO B STEP 1 ; action : END :`
`FOR I = A TO B STEP P ; action : END :`

2.8.4 Traduction HP48 ou HP49G mode RPN

A B FOR I *action* NEXT
A B FOR I *action* P STEP

2.8.5 Traduction HP49G mode Algébrique

FOR (I, A, B) *action* NEXT
FOR (I, A, B) *action* STEP P
L'instruction FOR déclare I comme variable locale et l'initialise automatiquement.

2.8.6 Traduction TI80 TI82 83+

:For (I,A,B) : *action* : End
:For (I,A,B,P) : *action* : End

2.8.7 Traduction TI89 92

:For I,A,B : *action* : EndFor
:For I,A,B,P : *action* : EndFor

2.8.8 Traduction SHARP EL9600

A $\Rightarrow$ I
Label DPOUR
If I>B Goto FPOUR
action1
I + 1 $\Rightarrow$ I
Goto DPOUR
Label FPOUR

2.9 L'instruction "Repeter"

Repeter *action* jusqua *condition*

2.9.1 Traduction MuPAD

repeat *action* until *condition* end_repeat

2.9.2 Traduction Casio

Do : *action* : LpWhile *condition*

2.9.3 Traduction HP38 HP40G

DO *action* : UNTIL *condition* END :

2.9.4 Traduction HP48 ou HP49G mode RPN

DO *action* UNTIL *condition* END

2.9.5 Traduction HP49G mode Algébrique

DO *action* UNTIL *condition* END

2.9.6 Traduction TI80

:LBL 1
:*action* :IF *non condition* :THEN :GOTO 1
:END

2.9.7 Traduction TI82 83+

```
:Repeat condition : action : End
```

2.9.8 Traduction TI89 92

```
:Loop :action :If condition :Exit :EndLoop
```

2.9.9 Traduction SHARP EL9600

```
Label REPETER
action
If non condition Goto REPETER
```

2.10 L'instruction "Tant que"

```
Tant que condition faire action ftantque
```

2.10.1 Traduction MuPAD

`while` *condition* `do` *action* `end_while` Vous pouvez aussi ouvrir le menu `MuPAD` sous menu `Shapes` et séléctionner `while`.

2.10.2 Traduction Casio

```
While condition : action : WhileEnd :
```

2.10.3 Traduction HP38 HP40G

```
WHILE condition REPEAT action : END :
```

2.10.4 Traduction HP48 ou HP49G mode RPN

```
WHILE condition REPEAT action END
```

2.10.5 Traduction HP49G mode Algébrique

```
WHILE condition REPEAT action END
```

2.10.6 Traduction TI80

```
 :LBL 1
:IF condition :THEN :action
:GOTO 1
:END
```

2.10.7 Traduction TI82 83+

```
:While condition : action : End
```

2.10.8 Traduction TI89 92

```
:While condition : action : EndWhile
```

2.10.9 Traduction SHARP EL9600

```
Label DTANTQUE
If non condition Goto FTANTQUE
action
Goto DTANTQUE
Label FTANTQUE
```

2.11 Les conditions ou expressions booléennes

Une condition est une fonction qui a comme valeur un booléen, à savoir elle est soit `vraie` soit `fausse`.

2.11.1 Les opérateurs relationnels

Pour exprimer une condition simple on utilise les opérateurs :
`=` `>` `>` $\leq$ $\geq$ $\neq$
Attention pour les calculatrices HP l'égalité se traduit comme en langage `C` par :
`==`

2.11.2 Les opérateurs logiques

Pour traduire des conditions complexes, on utilise les opérateurs logiques :
`ou et non`
qui se traduisent sur les calculatrices et en `MuPAD` par :
`or and not`
Attention!!! Je ne les ai pas trouvés sur la `TI80` et il faut alors traduire :
`Si` *`condition1`* `et` *`condition2`* `alors` *`action1`* `sinon` *`action2`* `fsi`
par :
`:IF` *`condition1`* `:THEN :IF` *`condition2`* `:THEN`
`:`*`action1`* `:ELSE :`*`action2`* `:END`
`:ELSE :`*`action2`* `:END`
`Si` *`condition1`* `ou` *`condition2`* `alors` *`action1`* `sinon` *`action2`* `fsi`
par :
`:IF` *`condition1`* `:THEN`
`:`*`action1`* `:ELSE :IF` *`condition2`* `:THEN :`*`action1`*
`:ELSE :`*`action2`* `:END :END`
Chez `Sharp` il faut parenthéser et écrire (A) or (B) par exemple.

2.12 Les fonctions

Dans une fonction on ne fait pas de saisie de données : on utilise des paramètres qui seront initialisés lors de l'appel.
Dans une fonction on veut pouvoir réutiliser le résultat :
on n'utilise pas la commande `affichage` mais la commande `retourne`.
On écrit par exemple en algorithmique :

```
fonction addition(A,B)
retourne A+B
ffonction
```

Cela signifie que l'on peut utiliser la fonction dans une expression.
On ne peut pas écrire de fonctions avec les calculatrices `Casio SHARPEL9600 HP38 HP40G TI83+`.

2.12.1 Traduction MuPAD

On écrit par exemple en `MuPAD` (`retourne` se traduit par `return`) :

```
addition:=proc(a,b)
begin
return(a+b)
end_proc:
```

REMARQUE : `return` fait sortir immédiatement de la fonction.

2.12.2 Traduction HP48 ou HP49G mode RPN

Pour la `HP48` ou la `HP49G mode RPN`, on suppose que les arguments de la fonction sont mis sur la pile avant l'appel de la fonction.
Dans l'écriture de la fonction les arguments sont des variables locales qui seront initialisées par les élements mis

sur la pile. Le résultat de la fonction est alors mis sur la pile.
L'exemple se traduit par :
```
≪→ A B
 ≪ A B + ≫
≫
'ADDITION' STO
```
puis on met les valeurs de A et B au niveau 1 et 2 de la pile, aprés l'exécution d'`ADDITION` leur somme se trouvera au niveau 1 de la pile.
N.B.pour des fonctions simples, on peut éviter les variables locales par exemple ≪ + ≫ `'ADDITION' STO`

2.12.3 Traduction HP49G mode Algébrique

Pour la `HP49G` :
L'exemple se traduit par :
```
≪→ A B
 ≪ A + B ≫
≫
STO▷ ADDITION
```
Puis, on tape :

`ADDITION`(4, 5)

2.12.4 Traduction TI89 92

```
:addition(a,b)
:Func
:Return a+b
:EndFunc
```

2.13 Les listes

On utilise les { } pour délimiter une liste.
Attention!!! En algorithmique on a choisit cette notation car c'est celle qui est employée par les calculatrices...à ne pas confondre avec la notion d'ensemble en mathématiques : dans un ensemble l'ordre des éléments n'a pas d'importance mais dans une liste l'ordre est important... Par exemple {} désigne la liste vide et {1, 2, 3} est une liste de 3 éléments.
`append` sera utilisé pour concaténer 2 listes ou une liste et un élément ou un élément et une liste :
`{1, 2, 3}->TAB`
`append(TAB, 4) ->TAB` (maintenant `TAB` désigne {1, 2, 3, 4}
`TAB[2]` désigne le deuxième élément de `TAB` ici 2.

2.13.1 Traduction MuPAD

Une liste est une suite d'expressions entre un crochet ouvrant [et un crochet fermant].
`[ ]` désigne la liste vide.
Exemple :
`l :=[1,2,2,3]`
`nops(l)` renvoie le nombre d'élements de la liste `l`.
`l[1]` ou `op(l,1)` renvoie le premier élement de la liste `l` : les élements sont numérotés de 1 à `nops(l)`.
`op(l)` renvoie `1,2,2,3`
`append(l,4)` ajoute l'élement 4 à la fin de la liste `l`.
De plus les listes peuvent être concaténées avec `.` ([1,2].[2,3]=[1,2,2,3]).
`Attention` une suite d'expressions entre une accolade ouvrante { et une accolade fermante } désigne un ensemble.
Exemple :
`A :={a,b,c} B :={a,d}` alors `A union B` désigne `{a,b,c,d}`
`A intersect B` désigne `{a}`
`A minus B` désigne `{b,c}`

2.13.2 Traduction Casio

Les variables listes ont pour noms : `List 1, List 2, ... List 6`.
Pour travailler avec des listes, il faut donner au départ le nombre d'éléments de la liste avec la commande `Dim List` par exemple :
`10->Dim List 1` (`List 1` désigne alors une liste de 10 éléments nuls).
On peut utiliser les commandes suivantes :
`seq(i*i, i, 1, 10, 2)` désigne la liste des carrés des 5 premiers entiers impairs.
`List 1[i]` désigne le ième élément de la liste `List 1`.

2.13.3 Traduction HP38 HP40G

Ici les listes peuvent avoir des longueurs non définies à l'avance.
Les variables listes ont pour noms : `L0, L1, L2, ... L9`.
On utilise les { } pour délimiter une liste.
Par exemple {1, 2, 3} est une liste de 3 éléments.
Mais {} ne désigne pas la liste vide, il faut utiliser la commande :
`CLEAR L1` pour initialiser la liste `L1` à vide pour la `HP38G`) et
`SYSEVAL 259588` pour initialiser la liste `L0` à vide (ou `SYSEVAL 259589` pour initialiser `L1` à vide etc pour la (`HP40G`).
Voici quelques commandes utiles :
`MAKELIST(I*I, I , 1, 10, 2)` désigne la liste des carrés des 5 premiers entiers impairs (2 indique le pas de `I`).
`L1(I)` désigne le `I`ème élément de la liste.
`CONCAT (L1, {5})` désigne une liste ayant l'élément 5 en plus des éléments de la liste `L1`.
`AUGMENT` n'existe que sur la `HP40G` et concaténe deux listes ou une liste et un élément.

2.13.4 Traduction HP48 ou HP49G mode RPN

Ici les listes peuvent avoir des longueurs non définies à l'avance.
On utilise les {} pour délimiter une liste.
Par exemple {1 2 3} est une liste de 3 éléments et {} désigne la liste vide.
On obtient le `P`ième élément de `L` sur la pile avec :
`L P GET`
Si on veut modifier le `P`ième élément de `L` (par exemple le mettre à 0) on écrira :
`'L' P 0 PUT` ou `L P 0 PUT 'L' STO`
En effet `L P 0 PUT` renvoie sur la pile la liste modifiée alors que :
`'L' P 0 PUT` modifie la liste `L`.
Pour concaténer deux listes ou une liste et un élément on utilise le `+` ou on utilise la commande `AUGMENT`.

2.13.5 Traduction HP49G mode Algébrique

Pour la `HP49G` en mode algébrique, les listes peuvent avoir des longueurs non définies à l'avance.
On utilise les {} pour délimiter une liste.
Par exemple {1 2 3} est une liste de 3 éléments et {} désigne la liste vide.
On obtient le `P`ième élément de `L` sur la pile avec :
`L[P]` ou `L(P)` ou `GET (L, P )`
Si on veut modifier le `P`ième élément de `L` (par exemple le mettre à 0) on écrira :
`PUT(L, P, 0) STO▷ L`
ou
`PUT('L', P, 0)`
En effet `PUT(L, P, 0)` renvoie la liste modifiée (sans modifier `L`) alors que :
`PUT ('L', P, 0)` modifie la liste `L`.
Pour concaténer deux listes ou une liste et un élément on utilise le `+` (et pour ajouter deux listes de mêmes longueurs on utilise la commande `ADD`) ou on utilise la commande `AUGMENT`..
La commade `SEQ` permet de constituer une liste, on tape :

$$\texttt{SEQ}(\texttt{'X*X'},\texttt{'X'},4,10,1)$$

on obtient :

$$\{16, 25, 36, 49, 64, 81, 100\}$$

2.13.6 Traduction TI80

Les variables listes ont comme noms `L1, L2, L3, L4, L5, L6`.
La commande `DIM` permet de créer une nouvelle liste (faite de zéros) ou de redimensionner une liste existante (numéro 3 du menu `OPS` de la touche `2nd STAT (LIST)`), par exemple :
`N` $->$ `DIM L1` : il y a deux possibilités,
soit L_1 n'existait pas alors L_1 est créée et contient une liste de zéros de longueur `N`,
soit L_1 existait, cette commande donne alors à L_1 la dimension `N` (soit en supprimant des termes, soit en rajoutant des zéros).
La commande `SEQ` permet de générer une liste (numéro 4 du menu `OPS` de la touche `2nd STAT (LIST)`) :
`SEQ`$(X^2, X, 0, 10, 2)->$ `L1` va par exemple créer la liste :
{0, 4, 16, 36, 64, 100} c'est à dire la liste des carrés de 0 à 10 avec un pas de 2 que l'on stocke dans `L1`.

2.13.7 Traduction TI83+

Les variables listes ont des noms prédéfinis $L_1, L_2, L_3, L_4, L_5, L_6$ ou bien peuvent avoir un nom de 1 à 5 caractères mais alors il faudra faire précéder ce nom du symbole $\llcorner$ (que l'on trouve dans `2nd LIST OPS B`) pour désigner cette variable.
Mais {} ne désigne pas la liste vide, il faut utiliser la commande :
`ClrList`L_1 (en position 4 du menu `EDIT` de la touche `STAT`) vide la liste L_1 (attention!!! dans un programme cela n'initialise pas la liste L_1 à vide).
La commande `dim` permet de créer une nouvelle liste (faite de zéros) ou de redimensionner une liste existante (en position 3 du menu `OPS` de la touche `2nd STAT (LIST)`) :
`N` $->$ `dim`(L_1) : il y a deux possibilités,
soit L_1 n'existait pas alors L_1 est créée et contient une liste de zéros de longueur `N`,
soit L_1 existait, cette commande donne alors à L_1 la dimension `N` (soit en supprimant des termes, soit en rajoutant des zéros).
La commande `seq` permet de générer une liste (en position 5 du menu `OPS` de la touche `2nd STAT (LIST)`) :
`seq`$(X^2, X, 0, 10, 2)->$ `TAB` va par exemple créer la liste :
{0, 4, 16, 36, 64, 100} c'est à dire la liste des carrés de 0 à 10 avec un pas de 2 que l'on stocke dans `TAB`.
$\llcorner$`TAB`(2) désigne le deuxième élément de `TAB` ici 4.
On peut aussi écrire :
$2->\llcorner$`TAB`(2)
La liste `TAB` est alors {0, 2, 16, 36, 64, 100}
ou si L_1 est de longueur `N` on peut rajouter un élément (par exemple 25) à L_1 en écrivant :
$25->L_1(N+1)$
`augment` permet de concaténer deux listes (en position 9 du menu `OPS` de la touche `2nd STAT (LIST)`).

2.13.8 Traduction TI89-92

`augment` permet de concaténer deux listes.
{} désigne la liste vide. Pour travailler avec des listes, on peut initialiser une liste de `n` éléments avec la commande `newlist`, par exemple :
`newlist(10)->L` (`L` est alors une liste de 10 éléments nuls).
On peut utiliser les commandes suivantes :
`seq(i*i, i, 1, 10)` qui désigne la liste des carrés des 10 premiers entiers, ou `seq(i*i, i, 0, 10, 2)` qui désigne la liste des carrés des 5 premiers entiers pairs (le pas est ici égal à 2).
Exemple :
`seq(i * i, i, 0, 10, 2)` $->$ `L` va par exemple créer la liste :
{0, 4, 16, 36, 64, 100} c'est à dire la liste des carrés de 0 à 10 avec un pas de 2 que l'on stocke dans `L`.
`L[i]` qui désigne le ième élément de la liste `L`.
On peut aussi écrire :
`2->L[2]`
La liste `L` est alors {0, 2, 16, 36, 64, 100}

ou si L est de longueur n on peut rajouter un élément (par exemple 121) à L en écrivant :
`121 -> L[n+1]`
Dans l'exemple précédent `n=6` on peut donc écrire :
`121 -> L[7]` (L est alors égale à {0, 2, 16, 36, 64, 100, 121}).
`left (L, 5)` désigne les 5 premiers éléments de la liste L.

2.13.9 Traduction SHARP EL9600

Ici les listes peuvent avoir des longueurs non définies à l'avance.
Les variables listes ont pour noms : `L1, L2, L3...L6`.
On utilise les { } pour délimiter une liste.
Par exemple {1, 2, 3} est une liste de 3 éléments.
Mais {} ne désigne pas la liste vide, il faut utiliser la commande :
`ClrList L1` (en position 4 dansle menu `OPE` de `STAT`) qui vide la liste `L1` (attention!!! dans un programme cela n'initialise pas la liste `L1` à vide).
Voici quelques commandes utiles :
`seq` qui se trouve en position 5 dans le sous menu `OPE` de `2nd` × (`LIST`) :
`seq(X*X, 1, 10, 2)` qui désigne la liste des carrés des 5 premiers entiers impairs (2 indique le pas de la variable qui est toujours `X`).
`L1(I)` désigne le Ième élément de la liste.
`augment(L1, {5})` désigne une liste ayant l'élément 5 en plus des éléments de la liste `L1` (`augment` se trouve en position 8 dans le sous menu `OPE` de `2nd` × (`LIST`).
La commande `dim` permet de créer une nouvelle liste (faite de zéros) ou de redimensionner une liste existante :
`N ->dim(L1)` crée `L1` une liste de zéros de longueur `N`.
mais si `L1` existait, cette commande donne à `L1` la dimension `N` (soit en supprimant des termes, soit en rajoutant des zéros).
Si `L1` est de longueur `N` on peut rajouter un élément (par exemple 25) à `L1` en écrivant :
`25 -> L1(N+1)`

3 Exemple : le PGCD par l'algorithme d'Euclide

Soient A et B deux entiers positifs dont on cherche le $PGCD$.
L'algorithme d'Euclide est basé sur la définition récursive du $PGCD$:

$$\begin{aligned} PGCD(A,0) &= A \\ PGCD(A,B) &= PGCD(B, A \mod B) \; si \; B \neq 0 \end{aligned}$$

où $A \mod B$ désigne le reste de la division euclidienne de A par B.
Voici la description de cet algorithme :
on effectue des divisions euclidiennes successives :

$$\begin{aligned} A = &\quad B \times Q_1 + R_1 \quad 0 \leq R_1 < B \\ B = &\quad R_1 \times Q_2 + R_2 \quad 0 \leq R_2 < R_1 \\ R_1 = &\quad R_2 \times Q_3 + R_3 \quad 0 \leq R_3 < R_2 \\ &..... \end{aligned}$$

Après un nombre fini d'étapes, il existe un entier n tel que : $R_n = 0$.
on a alors : $PGCD(A,B) = PGCD(B,R_1) = ... = PGCD(R_{n-1},R_n) = PGCD(R_{n-1},0) = R_{n-1}$

3.1 Traduction algorithmique

-Version itérative
Si B $\neq$ 0 on calcule R=A mod B, puis avec B dans le rôle de A (en mettant B dans A) et R dans le rôle de B (en mettant R dans B) on recommence jusqu'à ce que B=0, le PGCD est alors A.

```
Fonction PGCD(A,B)
Local R
```

```
tant que B ≠ 0 faire
  A mod B->R
  B->A
  R->B
ftantque
résultat A
ffonction
```

-Version récursive

On écrit simplement la définition récursive vue plus haut.

```
Fonction PGCD(A,B)
Si B ≠ 0 alors
  résultat PGCD(B,A mod B)
  sinon
  résultat A
fsi
ffonction
```

3.2 Traduction MuPAD

-Version itérative :

```
pgcd:=proc(a,b)
local r:
begin
While (b>0) do
r:=a mod b:
a:=b:
b:=r:
end_while:
return(a):
end_proc;
```

-Version récursive :

```
pgcd:=proc(a,b)
begin
if (b>0) then
 return(a)
 else
return(pgcd(b,a mod b)):
end_if:
end_proc;
```

3.3 Traduction Casio

-Version itérative :

On définit le programme INAB qui permet de rentrer deux nombres A et B :

```
‘‘A’’?->A
‘‘B’’?->B
ClrText
Return
```

Puis on tape le programme PGCD :

```
Prog ‘‘INAB’’
While B ≠ 0
  A-B*Intg(A/B)->R
  B->A
  R->B
WhileEnd
```

A△

-Pas de version récursive.

3.4 Traduction HP38G HP40G

-Version itérative

On écrit tout d'abord le sous-programme IN qui permet d'entrer deux nombres A et B :

```
INPUT A;"A";;;1:
INPUT B;"B";;;1:
ERASE:
```

puis on écrit le programme PGCD :

```
RUN IN:
DISP 3;"PGCD "{A,B}:
WHILE B ≠ 0 REPEAT
A MOD B ->R:
B ->A:
R ->B:
END:
DISP 4;"PGCD "A:
FREEZE:
```

-il n'y a pas de version récursive...mais on peut écrire le programme PGCDR :

```
DISP 3;"PGCD "{A,B}:
FREEZE:
IF B ≠ 0 THEN
A MOD B ->R:
B ->A:
R ->B:
PGCDR:
ELSE
DISP 3;"PGCD "A:
FREEZE:
END:
```

Il faut tout d'abord appeler le programme IN. Le programme PGCDR affiche les PGCD intermédiaires qui sont calculés. L'appel récursif PGCDR renvoie au programme PGCDR qu'il faut faire exécuter en appuyant sur le RUN du bandeau.

3.5 Traduction HP48 HP49G mode RPN

-Version itérative

```
≪ 0 → A B R
  ≪ WHILE B 0 ≠ REPEAT
       A B MOD 'R' STO
       B 'A' STO
       R 'B' STO
    END
    A
  ≫
≫
```

Puis on stocke ce programme dans `PGCD ('PGCD' STO)`.

-Version récursive

```
≪ 0 → A B
  ≪ IF B 0 ≠ THEN
       B A B MOD PGCDR
```

```
      ELSE
         A
      END
   ≫
≫
```
Puis on stocke ce programme dans `PGCDR ('PGCDR' STO)`.
N.B. il est possible de ne travailler qu'avec la pile sans variables locales en écrivant :
```
≪ IF DUP 0 ≠ THEN
   SWAP OVER MOD PGCDR
   ELSE DROP
   END ≫
```

3.6 Traduction HP49G mode Algébrique

-Version itérative
```
≪ → A,B
   ≪ 0 → R
      ≪ WHILE B ≠ 0 REPEAT
            A MOD B STO▷ R;
            B STO▷ A;
            R STO▷ B
         END;
         A
      ≫
   ≫
≫ STO▷ PGCD
```
Puis par exemple, `PGCD(45,75)` pour l'exécuter.

-Version récursive
```
≪ → A,B
   ≪ IF B ≠ 0 THEN
         PGCDR(B,A MOD B)
      ELSE
         A
      END
   ≫
≫ STO▷ PGCDR
```
Puis par exemple, `PGCDR(45,75)` pour l'exécuter.
Remarque :
Si on utilise la fonction du calcul symbolique `IREMAINDER` à la place de `MOD` dans les programmes précédents, `PGCD` (ou `PGCDR`) peut alors avoir comme paramètres des entiers de Gauss (c'est à dire les nombres $a + i \cdot b$ avec a et b entiers relatifs).

3.7 Traduction TI80

-Version itérative
```
PROGRAM :PGCD
:INPUT A
:INPUT B
:LBL 1
:IF B ≠ 0
:A-INT(A/B)*B->R
:B->A
:R->B
:GOTO 1
:END
:DISP A
```

-Pas de version récursive.

3.8 Traduction TI83+

-Version itérative

```
PROGRAM :PGCD
:Prompt A,B
:While B ≠ 0
:A-int(A/B)*B->R
:B->A
:R->B
:End
:A
```

-Pas de version récursive.

3.9 Traduction TI89 92

-Version itérative

```
:pgcd(a,b)
:Func
:Local r
:While b ≠ 0
:mod(a,b)->r
:b->a
:r->b
:EndWhile
:Return a
:EndFunc
```

-Version récursive

```
:pgcd(a,b)
:Func
:If b ≠ 0 Then
:Return  pgcd(b, mod(a,b))
:Else
:Return a
:EndIf
:EndFunc
```

3.10 Traduction SHARP EL9600

-Version itérative :

```
PGCD
Input A
Input B
Label DTQ
If B=0 Goto FTQ
A − int (A/B) ∗ B ⇒ R
B ⇒ A
R ⇒ B
Goto DTQ
Label FTQ
Print ”PGCD
Print A
End
```

-Pas de version récursive.

4 TABLE DES MATIÈRES

1– Pour commencer **1**
subsection1.1 Comment éditer et sauver un programme1
1.2 Comment corriger un programme 3
subsection1.3 Comment exécuter un programme5
1.4 Comment améliorer puis sauver sous un autre nom un programme 6
section2 Les différentes instructions7
2.1 Les commentaires 7
subsection2.2 Les variables8
2.3 Les Entrées 9
subsection2.4 Les Sorties9
2.5 La séquence d'instructions ou action 10
subsection2.6 L'instruction d'affectation11
2.7 Les instructions conditionnelles 11
subsection2.8 Les instructions "Pour" 13
2.9 L'instruction "Repeter" 13
subsection2.10L'instruction "Tant que"14
2.11 Les conditions ou expressions booléennes 15
subsection2.12Les fonctions15
2.13 Les listes 16
section3 Exemple : le PGCD par l'algorithme d'Euclide19
3.1 Traduction algorithmique 20
subsection3.2 Traduction MuPAD20
3.3 Traduction Casio 20
subsection3.4 Traduction HP38G HP40G21
3.5 Traduction HP48 HP49G mode RPN 21
subsection3.6 Traduction HP49G mode Algébrique22
3.7 Traduction TI80 22
subsection3.8 Traduction TI83+23
3.9 Traduction TI89 92 23
subsection3.10Traduction SHARP EL960023