

J.J. LABARTHE

BASIC 1000D

MISE A JOUR DU
MANUEL DE REFERENCE

1

Mise à jour du Basic 1000d

Les versions ST et TT

Il existe maintenant deux versions du Basic 1000d. La *version ST* est compatible avec tous les ordinateurs de la gamme Atari depuis le STF 520 jusqu'au TT. Elle fonctionne en couleur ou monochrome dans les 3 résolutions du ST. La *version TT*, entièrement compatible avec la version ST, tourne uniquement sur les modèles TT. Elle fonctionne dans toutes les résolutions du TT et gère pleinement les nouvelles possibilités (modes couleur/gris/barbouillis, ports série, extensions du gemdos et xbios). Elle utilise le coprocesseur et les possibilités supplémentaires du 68030. La table suivante compare les temps de calculs (en ms) de e^π avec 10 ou 1000 chiffres pour la version ST sur ST et TT et pour la version TT.

precision	v. ST sur ST	v. ST sur TT	v. TT
10	25	4,5	0,19
1000	86400	12600	3110

Nouveautés et compléments au manuel

REG_NUM

V_fonction Numéro de votre exemplaire du Basic et de votre facture d'achat

Exemple

L'instruction écrit sa valeur.

```
print reg_num
```

○ pages 22–5 (version ≥ 109)

Mots clefs

Les mots clefs tapés par [c] *lettre* sont écrits en majuscules ou minuscules suivant la touche CapsLock. La touche [c] – est l'inverse de [c] +.

[s] Help, [c] Help et [c] CR

Ces trois touches se saisissent du mot sous le curseur (désigné par μ). Elles ne sont actives que dans la fenêtre Edit Source.

[s] Help

Ouvre la fenêtre Help et cherche μ dans les titres du fichier Help.

[c] Help

Déplace le curseur sur le label μ . μ doit être le nom d'un label de la source.

[c] CR

Initialise la chaîne S avec μ et recherche la chaîne S dans toute la source.

○ *page* 25 (**version** ≥ 104)

Les touches [ca] \uparrow et [ca] \downarrow sont inactives sur TT. On peut maintenant utiliser [ca] A et [ca] B de façon équivalente.

○ *page* 38 (**version** ≥ 110)

Le dernier programme dans HELP doit être suivi de \T (comparer [s] Esc avec et sans ce \T). Le fichier HELP peut commencer par \A (supprimer la quatrième phrase en partant de la fin).

○ *page* 42 (**version** ≥ 110)

SAVE, IND (F17)

Cette commande du menu FILES sauvegarde la source sous forme d'un fichier ASCII avec indentation des lignes (voir aussi la note page 40–1).

○ *page* 110 (**version** ≥ 106)

Une progression peut maintenant être vide.

Exemple

La progression [4,1,1] est vide au lieu de se comporter comme (4).

```
forv v in [4,1,1]
  Ne passe pas ici
nextv
```

○ *pages* 120–1 (**version** ≥ 110)

Caractéristiques de l'écran**SCREEN_PX****SCREEN_PY**

Constantes Largeur et hauteur de l'écran en pixels

SCREEN_PC

Constante Nombre de plans couleurs

SCREEN_C

Constante Nombre de couleurs

SCREEN_CX**SCREEN_CY**

Constantes Largeur et hauteur d'un caractère en pixels

SCREEN_X

SCREEN_Y

Constantes Largeur et hauteur de l'écran en caractères

La largeur p_x et la hauteur p_y de l'écran en pixels sont données par `screen_px` et `screen_py`. Le nombre de plans de couleur p_c et le nombre de couleurs $c = 2^{p_c}$ s'obtiennent par `screen_pc` et `screen_c`. La taille $c_x * c_y$ en pixels des caractères est donnée par `screen_cx` et `screen_cy`. Le nombre de lignes y et le nombre de caractères par ligne x sont respectivement `screen_x` et `screen_y`. Les valeurs c_x , c_y , y et x se rapportent aux caractères affichés par l'éditeur et la commande `print`, d'autres tailles étant disponibles par les commandes graphiques.

Exemple

Le programme écrit une des lignes de la table page 120.

```
print"  r    y    x    c xmax ymax"
print resolution;
print justr$(screen_y,5);justr$(screen_x,5);
print using "#####";screen_c;screen_px;screen_py
```

Sortie (45 ms)

```
  r    y    x    c xmax ymax
  2    25    80    2   640  400
```

○ page 126, ligne -9

Changer 14 écrans en 6 écrans.

○ page 129

Exemple

La virgule terminale du premier `print` inhibe le changement de ligne. Elle diffère du point-virgule terminal par le saut de tabulation. Dans le dernier `print`, les virgules provoquent également des sauts de tabulation.

```
print 1,2,
print 3
print ,,4
```

Sortie (90 ms)

```
1                2                3
                4
```

○ page 136

La longueur de `screen$` est 38284.

○ page 152 (`version` ≥ 109)

\$281

Ce code est renvoyé par [c] -.

○ page 154

Pour que `music` ne s'arrête pas sur frappe d'une touche, il faut supprimer le retour sonore (bip-bip) par

```
pokeb $484,14
```

Le bip-bip se remet par

```
pokeb $484,15
```

○ page 186, ligne -9

Changer `dcondi` en `dcomp`.

○ pages 196-8

EXIT [*k*]

EXITIF [*k*]

EXITSELECT [*k*]

Formes supplémentaires des sorties de boucles et structures

k

entier*16 (par défaut $k = 1$)

On peut maintenant sortir de k boucles ou d'une imbrication de k structures IF ou SELECT.

Exemple

Comparer à l'exemple page 196. Après l'exécution de `exit 2`, le programme continue en dessous des deux `nextv`.

```
forv v in [1,2,1]
  forv u in (7/2,1,2)
    ift u=v exit 2
    print v;u
  nextv
nextv
```

Sortie

1 7/2

○ page 210

Changer deux fois ~ 30 en ~ 100 dans l'exemple.

○ page 213

Changer $m + 1024$ en $m + 1023$ dans la table.

○ page 233, ligne -7

Lire $a \geq 0$ au lieu de $a > 1$.

○ pages 237-41 (version ≥ 109)

La manipulation des dates et des calendriers a été très améliorée.

JULIANDATE

JULIANDATE(*mo,n,y*)

JULIANDATE(*c\$*)

V_fonction Jour Julien

mo

entier $\in [1, 12]$ (mois)

n

entier $\in [1, 31]$ (numéro du jour)

y
entier $\in [-32766, 32766]$ (année)

c\$
exprchaîne

La fonction **juliandate** renvoie le jour julien. On peut entrer la date à l'aide des 3 numéros du mois *mo*, du jour *n* et de l'année *y*. Pour *y*, une valeur négative comme $y = -124$ correspond à l'année 124 av. J.-C., et la valeur $y = 0$ est interprétée comme l'an 1 ($y = 1$). On peut aussi entrer la date à l'aide d'une chaîne. Par exemple **c\$="12/31/1991"** ou **c\$=" 8 / 17 / 124 BC "** pour le 17 août 124 av. J.-C. Pour la date du jour, on peut omettre les arguments.

Le calendrier est le calendrier grégorien à partir du 15 octobre 1582 et le calendrier julien avant cette date. Ainsi, le lendemain du jeudi 4 octobre 1582 (julien) est le vendredi 15 octobre 1582 (grégorien). Le jour julien calculé par la fonction **juliandate** est le nombre de jours écoulés depuis le 1 janvier 4713 av. J.-C.

DATE\$(j)

TIME_Y(j)

TIME_MO(j)

TIME_N(j)

TIME_D(j)

Formes supplémentaires des fonctions calendrier

j
entier $\in [-10247088, 13688960]$ (jour julien)

Ces nouvelles formes donnent la date d'un jour julien *j* quelconque et pas seulement du jour courant comme précédemment.

Exemple

Le 4 octobre 1582 était un jeudi et le 15 octobre 1582 un vendredi.

```
print time_d(juliandate("10/4/1582"))
print time_d(juliandate(10,15,1582))
```

Sortie (20 ms)

4
5

Exemple

La date dans 1000 jours.

```
j=juliandate+1000
print date$(j);time_n(j);time_mo(j);time_y(j)
```

Exemple

Le nombre de jours qui nous séparent du troisième millénaire.

```
print juliandate("1/1/2001")-juliandate
```

○ pages 244–6 (version ≥ 108)

Les deux nouvelles variables d'état suivantes facilitent le traitement des erreurs. En tant que commandes, la syntaxe est:

ERRORLABEL\$ [=] c

BREAKLABEL\$ [=] c

c

exprchaîne

1. Si *c* est vide ou contient le mot clef **STOP**, ces commandes sont identiques à (respectivement):

`on error stop`

`on break stop`

2. Si *c* contient le nom d'un label, ces commandes sont identiques à (respectivement):

`on error goto label`

`on break goto label`

3. Dans le cas de **breaklabel\$**, *c* peut contenir le mot clef **NEXT**. Par exemple

`breaklabel$ "next"`

équivalent à

`on break next`

Exemple

La fonction **binomial(a,b)** calcule un coefficient du binôme et écrit un message en cas d'erreur en se branchant en **binomial_err**. S'il n'y a pas d'erreur dans **binomial** la fonction ne modifie pas le traitement des erreurs du programme appelant. Cela est réalisé en sauvegardant la valeur d'entrée de **errorlabel\$** dans la pile puis en la rétablissant avant **return**.

Noter que les arguments de la fonction sont définis après la spécification du branchement d'erreur. L'instruction **errorlabel\$="binomial_err"** équivaut à: `on error goto binomial_err`.

```
print binomial(n,2)
print binomial(100,50)
print binomial(17)
stop
```

```
binomial:function
  push$ errorlabel$
  errorlabel$="binomial_err"
  function(a,b)
    ift not integerp(b) b=a-b
    value=ppwr(a,b)/ppwr(b)
    errorlabel$ pop$
    return
binomial_err:print
  print "erreur dans binomial"
```



```

        stop
Sortie (300 ms)
      1/2*n^2 -1/2*n
100891344545564193334812497256

```

erreur dans binomial

Exemple

La fonction II demande l'entrée d'un entier et répète sa demande en cas d'erreur. Elle ne modifie pas le traitement des erreurs du programme appelant.

```

      on error goto E1
      print sqr(II)
      on error goto E2
      print 1/II
      stop
E1:print "Erreur dans sqr"
      stop
E2:print "Erreur dans une division"
      stop
II:function
      push$ errorlabel$
II1:on error goto II2
      print "Entrer un entier"
      input value
      ift not integerp(value) goto II2
      errorlabel$ pop$
      return
II2:print "Erreur ! valeur non admise:";value
      goto II1

```

○ page 257 (version ≥ 106)

Si l'argument k est de la forme aaa(bbb), @k(etc) est remplacé par aaa(bbb,etc) au lieu de aaa(bbb)(etc).

Exemple

Comparer à l'exemple page 250.

```

      Sy sin
      Sy atn2(2)
      stop
Sy:print "@1(1)=";@1(1)
      return

```

Sortie (130 ms)

```

sin(1)= 0.8414709848~
atn2(2,1)= 0.1107148718~ E+1

```

○ page 358

Corriger l'équation (s) comme suit:

$$R = (-1)^i [w(Y_i) - f(Y_i)] \quad (s)$$

○ page 380

L'argument x de $\Gamma(x)$ peut être complexe, et pas seulement réel comme indiqué.

○ pages 402–4 (version ≥ 110)

XVCOLOR(i)

Variables d'état Couleur de numéro n

i

entier*16 $i \geq 0$

Les couleurs sont définies par 3 entiers R , V et B qui varient de 0 à 15 pour des proportions croissantes de rouge, vert et bleu. La couleur peut être codée par le nombre $RVB = 2^8 R + 2^4 V + B$. Un deuxième codage utilisé est compatible avec le codage *rvb* du ST décrit page 402. Les nombres r , v et b varient (STE ou TT) de 0 à 15, l'ordre des intensités croissantes étant 0, 8, 1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15. La relation entre R et r , par exemple, est donnée par $r = 8(R \bmod 2) + R \setminus 2$.

Les anciennes variables d'état `color` et `vcolor` utilisent le codage *rvb*. La nouvelle variable d'état `xvcolor` utilise le codage plus simple *RVB*.

La variable d'état `xvcolor(i)` fixe la couleur d'index i . L'index i va de 0 à `screen_c - 1` (ou à une autre valeur si la station de travail `v_h` \neq `v_h0`).

Exemple

Ecrit les valeurs de la couleur d'index 10 (basse résolution), sous les formes *rvb*, puis *RVB*. On a $r = v = 5$, $b = 13$, $R = V = 10$ et $B = 11$.

```
print/h/ vcolor($10);xvcolor($10)
```

Sortie (5 ms)

```
55D 0AAB
```

Sauvegardes d'écran

Nous vous renvoyons au fichier `HELP` pour un exemple commenté qui trace le graphe de la fonction $\Gamma(x)$ en résolution quelconque, puis sauvegarde l'écran sur disque.

La procédure `degascopy`, dont le nom rappelle `HARDCOPY` (voir page 125), permet de sauvegarder sur disque un écran au format `DEGAS` non compressé (PI1, PI2 ou PI3). La procédure `degasload` effectue la transformation inverse

en initialisant l'écran et la palette de couleurs à partir d'un fichier au format DEGAS.

2

Mise à jour du Basic 1000d–TT

Les versions ST et TT

Il existe deux versions du Basic 1000d. La *version ST*, compatible avec tous les ordinateurs de la gamme Atari depuis le STF 520 jusqu'au TT, fonctionne en couleur ou monochrome dans les 3 résolutions du ST. La *version TT*, uniquement pour les modèles TT, fonctionne dans toutes les résolutions et gère pleinement les nouvelles possibilités (modes couleur/gris/barbouillis, ports série, extensions du gemdos et xbios). Elle utilise le coprocesseur et les possibilités supplémentaires du 68030.

Tout programme écrit pour la version ST tourne avec la version TT (mais en général beaucoup plus vite). La commande `fractal` (tracé de fractales) et les variables d'état `aux_dev` (ports série), `xcolor`, `colorbank`, `graymode`, `color-mode` et `(no)smearmode` n'existent que dans la version TT. Cette brochure décrit ces nouvelles instructions et ajoute des compléments à quelques pages du manuel de référence du Basic 1000d.

La table suivante compare quelques temps de calculs (en ms). Le calcul `fsubs` substitue $x = \pi$ dans un polynôme de degré 100. Pour la version TT, les calculs flottants sont effectués directement par le coprocesseur jusqu'en `precision 15` et les opérations sur les grands nombres sont traitées par mots de 32 bits (au lieu de 16 bits).

		version ST sur ST	version ST sur TT	version TT sur TT
precision 10	e^π	25	4,5	0,19
	π^{1000}	14	2,8	0,25
	<code>ppwr</code> (π , 100)	171	35	0,84
	<code>fsubs</code>	202	40	12,40
precision 1000	e^π	86 400	12 600	3 110
	π^{1000}	8 600	1 050	390
En exact	1000!	4 800	725	650
	5000!	139 800	18 090	14 780
Factoriser $2^{67} - 1$	<code>prfact</code> \$	60 877 s%	12 365 s%	5 633 s
	<code>pollard</code>	695 s	159 s	91 s
	<code>brison</code>	411 s	99 s	81 s

Particularités de la version TT

Tracé de fractales

La commande **fractal** permet d'obtenir facilement de magnifiques effets graphiques. Elle n'existe que dans la version TT, le coprocesseur étant utilisé.

FRACTAL u, a, b [, K, x, y, w, h, Ic, ProgMach, M]

Commande Dessine une fractale

u, a, b

nombre complexes, domaine du paramètre de contrôle

K

entier, nombre d'itérations (par défaut $K = 100$)

x, y, w, h

entiers, rectangle de l'écran (par défaut tout l'écran)

Ic

nom de tableau ou de fonction, spécifie les couleurs

ProgMach

entier, fonction $f(z, p)$ en langage machine (par défaut $f(z, p) = z^2 + p$)

M

réel, limite (par défaut $M = 100$)

La commande **fractal** décore un ensemble de Mandelbrot. Elle ne peut être appelée qu'en **precision** ≤ 15 et avec le littéral complexe défini. A l'aide de ses nombreux arguments elle permet de créer rapidement une très grande variété d'images graphiques.

Indiquons d'abord brièvement le procédé utilisé. Il est tiré du livre de H O Peitgen et P H Richter *The Beauty of Fractals* (Springer-Verlag 1986), page 190, où nous vous renvoyons pour des explications plus détaillées.

On considère l'application $z \rightarrow f(z, p)$ qui à tout nombre complexe z associe le nombre complexe $f(z, p)$, où le nombre complexe p est un paramètre de contrôle. La commande **fractal** utilise l'application $z \rightarrow z^2 + p$, mais il est possible de définir d'autres applications à l'aide de l'argument **ProgMach**.

Les entiers x, y, w, h définissent le rectangle $ABCD$ de l'écran : le coin supérieur gauche A est donné par ses coordonnées absolues (x, y) , w est la largeur et h la hauteur (en pixels). A chaque pixel $M = (x + \alpha w, y + \beta h)$ (avec $0 \leq \alpha < 1, 0 \leq \beta < 1$) de ce rectangle est associée la valeur $p = u + \alpha a + \beta b$ du paramètre de contrôle. Ainsi, aux sommets du rectangle correspondent les valeurs $u, u + a, u + a + b$ et $u + b$. En prenant a réel et b imaginaire pur, l'axe réel est horizontal et l'axe imaginaire vertical.

Chaque pixel du rectangle est peint comme suit. On fixe p sur la valeur associée au pixel. On part de $z_0 = 0$ et on forme la suite $z_0, z_1 = f(z_0, p), z_2 = f(z_1, p), \dots, z_{k+1} = f(z_k, p), \dots$ On arrête le calcul à la première valeur

telle que $\text{cnorm}(z_k) = |z_k|^2$ devienne supérieur à M , sans toutefois dépasser la valeur $k = K$. Si $|z_k|^2 > M$, on prend la couleur de numéro $\text{Ic}(k)$ (où $0 < k \leq K$). Si $|z_0|^2 \leq M$, $|z_1|^2 \leq M$, \dots , $|z_K|^2 \leq M$, on prend la couleur de numéro $\text{Ic}(0)$.

Voici quelques indications supplémentaires sur les arguments à donner. Seuls u , a et b sont obligatoires. On peut ensuite se dispenser de donner les arguments à partir d'un rang quelconque. Pour l'application $z \rightarrow z^2 + p$, la partie intéressante est à l'intérieur du rectangle $u = -2.25 + 1.5i$, $a = 3$, $b = -3i$.

Il vaut mieux prendre K assez grand ($K = 200, 300, \dots$), mais le temps de calcul est pratiquement proportionnel à K .

Si on donne l'argument Ic , il faut définir un tableau ou une fonction ayant ce nom. L'expression $\text{Ic}(k)$ doit renvoyer un entier*32 pour $k = 0, 1, \dots, K$. La couleur utilisée sera celle de numéro $\text{mod}(\text{Ic}(k), \text{screen_c})$, où screen_c est le nombre de couleurs. Si l'argument Ic est omis, tout se passe comme si on prenait $\text{Ic}(k) = k$ pour $k \neq 0$ et $\text{Ic}(0) = -1$ qui donne (usuellement) la couleur noire. Lorsque $K \gg \text{screen_c}$, il est conseillé de donner la même couleur à des valeurs consécutives de k .

L'argument **ProgMach** permet de définir d'autres applications $z_x + iz_y \rightarrow f(z_x + iz_y, p_x + ip_y)$. La fonction $f(z, p)$ est typiquement calculée 10^8 fois au cours d'une commande **fractal**. Pour que **fractal** reste rapide (quelques minutes et non plusieurs heures), nous avons préféré imposer que la fonction $f(z, p)$ soit écrite en langage machine, et non en Basic. A l'adresse **ProgMach** il faut placer le mot long \$65100ADE (erreur Magic si ce mot est absent), suivi du sous programme de calcul de f . En entrée, les arguments de f sont dans les registres du coprocesseur : $\text{FP0} = z_x$, $\text{FP1} = z_y$, $\text{FP6} = p_x$ et $\text{FP7} = p_y$. En sortie, la valeur calculée de f doit être placée dans FP0 (partie réelle) et FP1 (partie imaginaire). La fonction peut détruire seulement les registres FP2 , FP3 , FP4 et D0 . Les autres registres FP5-FP7 , D1-D7 et A0-A7 doivent être conservés. Pour le dernier argument, la valeur $M = 100$ convient généralement.

Exemple

On trace 4 fractales sur le même écran. Les trois premières correspondent à la même fenêtre de valeurs du paramètre p . Comme la fenêtre est très étroite, de nouveaux détails apparaissent quand on passe de $K = 200$ à $K = 500$. Pour $K = 500$, le tracé qui regroupe les couleurs par blocs de 50 valeurs de k (à l'aide de la fonction **m50**) fait ressortir les spirales en résolution TT moyenne.

La fractale du coin droit inférieur utilise l'application $z \rightarrow z^3 + p$. Le programme machine, précédé du mot magique \$65100ADE, est placé dans la variable **cube\$**. Le temps d'exécution du programme dépend de la résolution (421 s en TT basse, 867 s en TT moyenne).

```
cube$=mkz$($65100ADEF2276A80F2000100F23C51800003F200060
OF2000923F2001223F2000A80F2000D23F20011A3F2001128F2000
EA8F20008A3F2001423F2001CA2F2001822F21F4A804E75,72)
complex i
```

```

cls
u=-0.7454356+0.1130137*i
a=0.000014
b=-0.00001*i
y1=4*screen_cy+16
h=(screen_py-y1-16)\2
w=(screen_px-16)\2
fractal u,a,b,200, 0, y1,w,h
fractal u,a,b,500,w+16, y1,w,h,m50
fractal u,a,b,500, 0,y1+h+16,w,h
fractal -1-1.5*i,2,3*i,100,w+16,y1+h+16,w,h,m, ptr(
cube$),1000
message "temps=\"%timer&" "
ift keyget
stop
m:function(index t)
value=t
ift t=0 value=-1
return
m50:function(index t)
value=t\50
ift t=0 value=-1
return

```

Enfin voici le programme en assembleur qui a été placé dans cube\$.

```

D.L $65100ADE ;mot long magique
FMOVE.X FP5,-(SP) ;FP5 doit être conservé
FMOVE FP0,FP2 ;=  $z_x$ 
FMOVE.W #3,FP3 ;= 3
FMOVE FP1,FP4 ;=  $z_y$ 
FMUL FP2,FP2 ;=  $z_x^2$ 
FMUL FP4,FP4 ;=  $z_y^2$ 
FMOVE FP2,FP5 ;=  $z_x^2$ 
FMUL FP3,FP2 ;=  $3z_x^2$ 
FMUL FP4,FP3 ;=  $3z_y^2$ 
FSUB FP4,FP2 ;=  $3z_x^2 - z_y^2$ 
FSUB FP3,FP5 ;=  $z_x^2 - 3z_y^2$ 
FMUL FP2,FP1 ;=  $3z_x^2 z_y - z_y^3$ 
FMUL FP5,FP0 ;=  $z_x^3 - 3z_x z_y^2$ 
FADD FP7,FP1 ;=  $\text{Im}(z^3 + p)$ 
FADD FP6,FP0 ;=  $\text{Re}(z^3 + p)$ 
FMOVE.X (SP)+,FP5
RTS

```


Caractéristiques de l'écran

Les constantes suivantes et la variable **resolution** existent aussi dans la version ST. Dans la version TT, pour les nouvelles résolutions du TT, de nouvelles valeurs apparaissent.

RESOLUTION0

Constante Résolution du bureau

RESOLUTION r

Variable d'état Résolution

r

entier*16

SCREEN_PX

SCREEN_PY

Constantes Largeur et hauteur de l'écran en pixels

SCREEN_PC

Constante Nombre de plans couleurs

SCREEN_C

Constante Nombre de couleurs

SCREEN_CX

SCREEN_CY

Constantes Largeur et hauteur d'un caractère en pixels

SCREEN_X

SCREEN_Y

Constantes Largeur et hauteur de l'écran en caractères

La constante **resolution0** a la valeur r_0 dépendant de la résolution au lancement du Basic 1000d. La largeur p_x et la hauteur p_y de l'écran en pixels sont données par **screen_px** et **screen_py**. Le nombre de plans de couleur p_c et le nombre de couleurs $c = 2^{p_c}$ s'obtiennent par **screen_pc** et **screen_c**. La taille $c_x * c_y$ en pixels des caractères est donnée par **screen_cx** et **screen_cy**. Le nombre de lignes y et le nombre de caractères par ligne x sont respectivement **screen_x** et **screen_y**.

Les valeurs c_x , c_y , y et x se rapportent aux caractères affichés par l'éditeur et la commande **print**, d'autres tailles étant disponibles par les commandes graphiques.

La variable d'état **resolution** se comporte comme décrit page 120 pour les résolutions basse et moyenne du ST. Pour les autres résolutions **resolution** permet de changer la fonte système. Il y a 3 possibilités correspondant aux fontes 8*16, 8*8 et 16*32. On peut parcourir ces fontes en tapant [s] Tab. Noter que les valeurs des "constantes" **screen_cx**, **screen_cy**, etc. sont alors modifiées.

	r_0	p_x	p_y	p_c	c	r	c_x	c_y	x	y
ST basse	0(1)	320	200	4	16	0	8	8	40	25
ST moyenne	1(0)	640	200	2	4	1	8	8	80	25
ST haute	2	640	400	1	2	2	8	16	80	25
						3	8	8	80	50
						4	16	32	40	12
TT basse	7	320	480	8	256	7	8	16	40	30
						8	8	8	40	60
						9	16	32	20	15
TT moyenne	4	640	480	4	16	4	8	16	80	30
						5	8	8	80	60
						6	16	32	40	15
TT haute	6	1280	960	1	2	6	16	32	80	30
						7	8	16	160	60
						8	8	8	160	120

Exemple

Le programme écrit une des lignes de la table.

```
print" r0  px  py  pc   c   r  cx  cy   x   y"
print resolution0;screen_px;screen_py;
print using "#####";screen_pc;screen_c;resolution;
print justr$(screen_cx,5);justr$(screen_cy,5);
print justr$(screen_x,5);justr$(screen_y,5)
```

Sortie (50 ms)

```
 r0  px  py  pc   c   r  cx  cy   x   y
  2 640 400   1   2   2   8 16   80  25
```

○ page 136

La longueur de `screen$` dépend de la résolution de l'écran ; en octets c'est 38284 (modes du ST) ou 175154 (modes du TT).

○ pages 170–1

AUX_DEV [k]

Variable d'état Fixe/lit le port série

k

entier*16 (par défaut $k = 6$)

Les ports série du TT sont numérotés de 6 à 9.

<i>k</i>	Port série
6	Modem 1 compatible ST
7	Modem 2
8	Serial 1
9	Serial 2

Pour travailler sur le port “Serial 2”, par exemple, on fixe d’abord `aux_dev` sur 9 :

```
aux_dev 9
```

puis on utilise les commandes usuelles de l’interface série comme :

```
open "o",#n,"aux:"
print #n,"..."
```

ou aussi `inp(1)` et `out 1 {,b}` (voir page 175).

Pour plus de détails, et aussi pour le problème de la configuration des ports série, voir ST Mag, 55, 84-7.

Exemple

L’appel `inp?` teste si un caractère est disponible sur le port Modem 2.

```
aux_dev 7
print aux_dev
print inp?(1)
aux_dev
print aux_dev
```

○ pages 375-8

Voici un autre exemple de calculs géométriques (version ST ou TT).

Théorème de l’arc brisé (Archimède)

Soient A, B, M et P des points sur un demi-cercle, M étant le milieu de l’arc AB , et P un point arbitraire de l’arc MB . Soit H la projection orthogonale de M sur la droite AP . Le théorème affirme que

$$AH = HP + PB. \tag{1}$$

Pour démontrer cette équation avec le Basic 1000d nous devons d’abord la transformer de sorte que seulement les carrés des longueurs des segments interviennent (ce qui évite des racines carrées). Il est facile de montrer que l’équation (1) est équivalente aux équations

$$AH^2 = HP^2 + PB^2 + 2\,HP \times PB \tag{2}$$

$$(AH^2 - HP^2 - PB^2)^2 = 4\,HP^2 \times PB^2. \tag{3}$$

Le programme suivant démontre l’équation (3). Pour les notations, nous vous renvoyons pages 375-8. Les coopoly utilisées sont $A = (-c, s, 1)$, $B = (c, s, 1)$, $M = (0, 1, 1)$, $P = (C, S, 1)$ et $H = (h_x, h_y, h_z)$. On a pris le rayon du demi-cercle

égal à 1. Les nombres s et c sont liés par $s^2 + c^2 = 1$ et S et C par $S^2 + C^2 = 1$. Les coopoly de H sont calculées par la procédure **projorth** de la bibliothèque MATH (qui doit être chargée). Les carrés des longueurs des segments AH , HP et PB sont mis dans **AH2**, **HP2** et **PB2** par la fonction **dist2** de la bibliothèque MATH. En sortie on obtient $Z = 0$ qui montre le théorème.

```
cond s^2+c^2-1,s
cond S^2+C^2-1,S
var hx,hy,hz
projorth -c,s,1, C,S,1, 0,1,1, hx,hy,hz
AH2=d2(-c,s,1, hx,hy,hz)
HP2=d2(hx,hy,hz, C,S,1)
PB2=d2(C,S,1, c,s,1)
Z=(AH2-HP2-PB2)^2-4*HP2*PB2
print Z
stop
d2:function(a,b,c, d,e,f)
  local var g,h
  dist2(a,b,c, d,e,f, g,h)
  value=g/h
  return
```

Sortie (ST 7885 ms, TT 1745 ms)

0

○ pages 402-4

COLOR(n)

VCOLOR(i)

XCOLOR(N)

XVCOLOR(i)

Variables d'état Couleur de numéro n (N) ou d'index i (**xcolor** pour la version TT seulement, **color**, **vcolor** et **xvcolor** aussi pour la version ST)

n, N, i

entiers*16 $n \in [0, 15]$, $N \in [0, 255]$, $i \geq 0$

Les couleurs sont définies par 3 entiers R , V et B qui varient de 0 à 15 pour des proportions croissantes de rouge, vert et bleu. La couleur peut être codée par le nombre $RVB = 2^8R + 2^4V + B$. Un deuxième codage utilisé est compatible avec le codage *rvb* du ST décrit page 402. Les nombres r , v et b varient (STE ou TT) de 0 à 15, l'ordre des intensités croissantes étant 0, 8, 1, 9, 2, 10, 3, 11, 4, 12, 5, 13, 6, 14, 7, 15. La relation entre R et r , par exemple, est donnée par $r = 8(R \bmod 2) + R \setminus 2$.

Les variables d'état **color** et **vcolor** utilisent le codage *rvb*. Les variables d'état **xcolor** et **xvcolor** utilisent le codage plus simple *RVB*.

La variable d'état **color(n)** ($n \in [0, 15]$) lit ou fixe la couleur de l'écran de numéro n de la palette compatible ST via la fonction **xbios(7)**.

La variable d'état `xcolor(N)` ($N \in [0, 255]$) lit ou fixe la couleur de l'écran de numéro N de la palette du TT via la fonction `xbios(83)`. Elle n'est disponible que dans la version TT.

Les variables d'état `vcolor(i)` et `xvcolor(i)` lisent ou fixent la couleur d'index i via les fonctions `vdi(14)` et `vdi(26)`. L'index i va de 0 à `screen_c - 1` (ou à une autre valeur si la station de travail `v_h` \neq `v_h0`).

Sur TT, dans la résolution ST haute (ou duochrome), les 2 couleurs ont les numéros $N = 255$ et 254 , indépendants de `colorbank`, et correspondent aux index $i = 0$ et 1 . En modifiant le bit 0 de `color(0)`, comme indiqué pages 402–3, ou le bit 1 de `xcolor(0)`, les couleurs du fond et du texte sont permutées (`xcolor(255)` et `xcolor(254)` sont inchangés, mais la correspondance index \leftrightarrow numéro est modifiée).

Dans la résolution TT haute (monochrome), ces variables n'ont aucun effet sur l'écran.

Exemple

Détermine les index actifs, puis la correspondance entre index et numéro (elle dépend de `colorbank`). On obtient des résultats analogues aux tables page 403. Sur TT monochrome, `vcolor(0)` et `vcolor(1)` ne prennent que les valeurs 0 et \$FFF et ce programme ne fonctionne pas.

```
colorbank random(16)
print "index(VDI)    numéro(XBIOS) pour colorbank=";colo
rbank
for n=0,255
  M=xvcolor(n)
  Z=$770
  ift M=Z Z=$27
  xvcolor(n)=Z
  if xvcolor(n)=Z
    print justr$(n,5);
    for j=0,255
      i=modr(n+j,256)
      if xcolor(i)=Z
        xvcolor(n)=Z+2
        if xcolor(i)=Z+2
          print justr$(i,15)
          exit
        endif
        xvcolor(n)=Z
      endif
    next j
    ift j=256 print justr$("?",15)
  endif
  xvcolor(n)=M
```

```
next n
colorbank 0
```

COLORBANK k

Variable d'état Banque de couleur

k

entier*16 $k \in [0, 15]$

Dans les résolutions ST basse, ST moyenne et TT moyenne, la banque k correspond aux couleurs de numéros $16k$ à $16k + 15$ de la palette TT. C'est cette banque qui est utilisée comme palette ST. $u = \text{color}(n)$ et $U = \text{xcolor}(16k + n)$ sont alors reliés par $U = 2(u \text{ and } \$777) + (u \text{ and } \$888)/8$.

GRAYMODE [g]

COLORMODE [g]

Paire de variables d'état Mode gris ou couleur

g

entier*16

Ces variables prennent les valeurs 0 (mode couleur) ou 1 (mode gris). En mode gris la partie *VB* du code *RVB* des couleurs est interprétée comme une nuance de gris (0-255).

En tant que commandes, **graymode**, **graymode g** ou **colormode g** avec g impair mettent le mode gris. De même, **colormode**, **graymode g** ou **colormode g** avec g pair mettent le mode couleur.

Exemple

En sortie, le mode gris/couleur d'entrée est remis. Comme **random(2)** prend les valeurs 0 ou 1, ce mode varie aléatoirement dans la boucle.

```
cls
push graymode
for i=0,15
  for j=0,15
    f_color random(screen_c)
    pbox 190+8*j,80+8*i,197+8*j,87+8*i
  next j
  graymode random(2)
  print graymode
next i
graymode pop
```

SMEARMODE [s]

NOSMEARMODE [s]

Paire de variables d'état Mode barbouillis

s

entier*16

Ces variables prennent les valeurs 0 (mode normal) ou 1 (mode barbouillis). En mode barbouillis la couleur d'un point contamine la couleur des points situés à sa droite. Il n'y a pas d'effet en monochrome.

En tant que commandes, **smearmode**, **smearmode s** et **nosmearmode s**, avec *s* impair mettent le mode barbouillis. Inversement, **nosmearmode**, **smearmode s** et **nosmearmode s**, avec *s* pair passent en mode normal.

Exemple

Noter que le débogueur ne passe pas en mode barbouillis. Une autre façon de revenir en mode normal consiste à ouvrir la fenêtre DEBUG par F9, F19 ou F29 et à appuyer sur A.

```
t_height 13
for i=0,15
  t_color i
  text 300,64+16*i,conc$(j=0,10 of chr$(65+random(26)))
next i
graymode
pause 1000
smearmode
pause 1000
colormode
nosmearmode
```

○ *pages* 406, 409, 415 et 419

L'index de couleur *i* dans les variables d'état **l_color**, **f_color**, **m_color** et **t_color** n'est pas limité à [0,15], mais varie de 0 au nombre de couleurs disponibles moins 1 (**screen_c** - 1 pour l'écran). En fait, si on donne une valeur illégale, l'index 1 est sélectionné.

○ *pages* 422-30

Deux nouveaux appels du GEMDOS sont possibles. Compléter la table avec:

14 l, l

Informar le GEMDOS de la présence de mémoire alternative.

44 l, w

Allouer une zone mémoire (avec préférences).

○ *pages* 433-5

De nouvelles fonctions XBIOS sont possibles. Compléter la table avec:

29 w, w

Lire/fixer l'attente lors d'un changement de piste du lecteur A ou B.

2A l, w, l, w

Lire des secteurs (ACSI et SCSI).

2B l, w, l, w

Ecrire des secteurs (ACSI et SCSI).

2C w

Analogue à `aux_dev`.

2E w, w, w, l

Lire/écrire la mémoire non volatile (NVM) (50 octets sur TT).

50 w

Fixe le registre de mode écran (shiftmode).

51

Lit le registre de mode écran (shiftmode).

52 w

Analogue à `colorbank`.

53 w, w

Analogue à `xcolor`.

54 w, w, l

Fixe la palette TT.

55 w, w, l

Lit la palette TT.

56 w

Analogue à `graymode`.

57 w

Analogue à `smearmode`.