Syntaxe langage *draw++*

lien google doc : <https://docs.google.com/document/d/1Fm-62grZrXCt1T9LBf8nMlmtLsHy9HX8JFHtnub0rbk/edit>

Repère :

* point (0,0) : coin supérieur gauche de la fenêtre d’affichage.
* Axe X : croissant vers la droite
* Axe Y : croissant vers le bas

Probleme :

* comment récupérer le nom d’un curseur ? (ex: une boucle appelant plusieurs curseurs)

| **Instructions élémentaires** | **Explications** |
| --- | --- |
| **CursorCreate**<*name*>; | Crée un curseur de nom *name*. |
| **CursorColor**<*name*, *color*>; | Attribue la couleur *color* d’écriture au curseur de nom *name*.  Couleur par défaut du curseur : *auto* (noir).  La couleur peut-être donnée au format RGB (*rgb(255,255,255)*) ou par un nom de couleur (*dark*, *blue*, *orange*). |
| **CursorThick**<*name*, *nb*>; | Attribue l’épaisseur du trait *nb* au curseur de nom *name*.  *nb* est de type float.  Épaisseur par défaut du curseur : 1. |
|  |  |
| **CursorPosition**<*length*>;  ou  **CursorPosition**<*x*, *y*>;  ou  **CursorPosition**<**r** *x*, **r** *y*>; | Déplace le curseur :   * de *length* dans le sens du curseur * absolument au point de coordonnées (x,y). * relativement à sa position initiale de *x* en abscisse et *y* en ordonnée. |
| **CursorRotate**<*name*, *degree*>;  ou  **CursorRotate**<*name*, **r** *degree*>; | Tourne le curseur de nom *name* de *degree* degrés dans le sens horaire. Cette rotation peut être absolu, dans ce cas 0° correspondra au curseur dirigé vers le haut.  Cette rotation peut être relative, dans ce cas CursorRotate(*name*, r 10) revient à ajouter 10° supplémentaire à l’inclinaison du curseur.  Par défaut le pointeur est dirigé vers le haut (0°). |
| **Line**<*name*, *length*>;  ou  **Line**<*name*, *x*, *y*>;  ou  **Line**<*name*, **r** *x*, **r** *y*>; | Trace une ligne :   * en avançant de *length* dans la direction du curseur * en partant de la position du curseur vers le point de coordonnées (x,y) (déplacement absolue) * en avançant de *x* en abscisse et *y* en ordonnée à partir de la position du curseur (déplacement relatif) |
| **Square**<*name*, *length\_side*)>; | Trace un carré de côté *length\_side*, le curseur fait une rotation de 90° dans le sens horaire après avoir fini de tracer chaque côté.  A noter : le curseur revient dans le même sens que celui avant exécution de la fonction. |
| **Circle**<*name*, *diameter*>; | Trace un cercle de diamètre *diameter*.  Le curseur se trouve sur l’arc et le centre à une distance *diameter* du curseur.  Le tracé se fait dans le sens horaire et le curseur retrouve son sens initial. |
| **Point**<*name*, *color, size*, *name\_pt*>; | Trace un point de couleur *color*, de diamètre *size* et affiche son nom *name\_pt*.  Le curseur ne change ni de position, ni de sens. |
| **CircleArc**<*name*, *diameter*, *angle*>; | Trace un arc-de-cercle de diamètre *diameter*.  Le curseur se trouve sur l’arc et le centre à une distance *diameter* du curseur.  Le tracé se fait dans le sens horaire pour *angle* > 0, anti-horaire pour *angle* < 0.  le sens du curseur change |
|  |  |
| Animer un dessin ????? |  |

**Instructions évoluées :**

1. **Variables**

Chaque variable est définie par son type :

* int
* double
* float
* char
* bool

Format d’affectation de variable : *type nom\_variable = valeur ;*

Les espaces autour du signe ‘=’ et avant le ‘;’ sont optionnels.

Exemples :

*bool a= TRUE*

*bool b= FALSE;*

*int x = 5 ;*

*float y = 5.24;*

*double z =5.1234567891011 ;*

*char c1=‘c’;*

*char c2 = “b”;*

1. **Instruction de bloc**
   1. **Parenthèses prioritaires**

Les parenthèses ont un rôle prioritaire dans l’exécution d’une ligne de commande.

Les calculs comportant des parenthèses calculeront dans un premier lieu le contenu des parenthèses.

Exemple:

*a = 5\*(2+8);*

*// ici 2+8 sera calculé, la multiplication est faite dans un second temps*

* 1. **Fonctions & bloc d’instructions**

Des fonctions peuvent être déclarées par l’utilisateur de la façon suivante :

fct nom\_fonction<int var\_a, double var\_b, char var\_c>{

int x = 2\*a+1;

return x;

}

*fct* : permet de déclarer une fonction

< … > : contient les paramètres de la fonction (munie de leurs types). Les paramètres sont séparés par une virgule.

{ … } : contient le bloc d’instruction de la fonction. l’accolade ouvrante est sur la ligne de déclaration de la fonction et la fermante est sous la dernière ligne du bloc.

L’indentation dans le bloc d’instructions est conseillée mais n’est pas obligatoire.

1. **Comparateurs**

Voici un tableau des comparateurs à utiliser avec les instructions conditionnelles :

| **Comparateurs** | **Significations** |
| --- | --- |
| AND | et |
| OR | ou |
| NOT(...) | Négation de la proposition entre parenthèses. Si la proposition entre parenthèses est vraie, elle deviendra fausse (et inversement) |
| == | est égal à |
| != | est différent de |
| > | supérieur strictement à |
| >= | supérieur ou égale à |
| < | inférieur strictement à |
| <= | inférieur ou égale à |

1. **Instructions conditionnelles** (if, else if, else)

La condition est toujours donnée entre parenthèses après la commande *if*.

Les accolades ouvrantes et fermantes délimitent l’instruction à exécuter.

Exemple :

if (a==TRUE AND b>1){

…

}

else if (a==FALSE AND b>1){

…

}

else {

…

}

1. **Instructions répétitives**
2. Boucle for

La boucle *for* permet de répéter une instruction un certain nombre de fois à l’aide d’une variable qui incrémente à chaque tour de boucle.

Syntaxe :

for (int i = *valeur\_de\_depart*, i<*valeur\_max*, *incrementation*){

…

}

Exemple :

for (int i = 0, i<3, i=i+1){

…

}

1. Boucle while

La boucle *while* permet de répéter une instruction un certain nombre de fois à l’aide d’une variable initialisée avant la boucle. La boucle est exécutée tant que la condition entre parenthèses est toujours valide.

Syntaxe :

while (*condition*){

…

}

Exemple :

int i = 0;

while (i<3){

…

}

1. Boucle do-while

La boucle *do-while* permet de répéter une instruction un certain nombre de fois à l’aide d’une variable initialisée avant la boucle. La boucle est exécutée une première fois et continue de s’exécuter tant que la condition entre parenthèses reste valide.

Syntaxe :

do-while (*condition*){

…

}

Exemple :

int i = 0;

do-while (i < -1){

…

}

// le contenu de la boucle est exécuté 1 seule fois

int j = 0;

do-while (j < 3){

…

j=j+3;

}

// le contenu de la boucle est exécuté 3 fois