Projet de programmation C

Vector Text-based Editor

Halim Djerroud, Fabien Calcado, Asma Gabis

Mars 2023

Consignes & Informations générales :

➔ Ce projet est à réaliser exclusivement en langage C

➔ Pièce jointe au projet :

— Un exécutable (version pour Windows) pour vous donner une idée du fonctionnement attendu et de certaines fonctionnalités souhaitées.

➔ Organisation des équipes :

— Ce projet est à réaliser en binôme (un seul trinôme est autorisé si un nombre impair d'élèves)

— La liste des équipes est à remettre aux enseignants **au plus tard** au début de la première séance de suivi de projet

➔ Dépôt du projet : Deux dépôts sont exigés :

— Un dépôt intermédiaire pour la partie 1 du projet

— Un dépôt final de l'ensemble du projet avant la soutenance

➔ Dates clés :

— Date de présentation du projet : **13/03/2023**

— Date de publication partie 1 : **18/03/2023**

— Date de séance de suivie 1 : semaine du **20/03/2023**

— Date de publication partie 2 : **XX/03/2023**

— Date de séance de suivie 2 : semaine du **17/04/2023**

— Date du dépôt intermédiaire : **23/04/2023 à 23h59**

— Date de séance de suivie 3 : semaine du **15/05/2023**

— Date du dépôt final : **21/05/2023 à 23h59**

— Date de soutenance : Semaine du **22/05/2023**

➔ Rendu final : S'effectue sur Moodle sous forme d'une archive **.zip** contenant :

— Le code du projet avec tous les fichiers **.h** et **.c**

— Le rapport en **.pdf**

— Un fichier **README.txt** donnant la liste des programmes et comment les utiliser en pratique. Ce fichier doit contenir toutes les instructions nécessaires à l'exécution pour expliquer à l'utilisateur comment compiler et se servir de votre logiciel.

➔ Évaluation

— Barème détaillé : sera fourni plus tard

— Note finale du projet = Note code + Note rapport + Note soutenance

— **Rappel :** note projet = 15% de la note du cours « Algorithmique et Structures de données 1 »

— Les membres d'une même équipe peuvent avoir des notes différentes en fonction des efforts fournis dans la réalisation de ce projet.

Table des matières

1. Le dessin vectoriel **5**
- 1.1. Principe de base . . . 5
- 1.2. Exemples d'interface de logiciel de dessin vectoriel . . . 6

2. Objectifs du projet **8**
- 2.1. Représentation mémoire des données . . . 9
- 2.2. Interface en ligne de commande . . . 9
- 2.3. Génération du dessin à l'écran . . . 10

I. Structures de données des formes **12**

3. Les structures spécifiques des formes disponibles **12**
- 3.1. Structure `Point` . . . 12
- 3.2. Structure `Line` . . . 13
- 3.3. Structure `Square` . . . 14
- 3.4. Structure `Rectangle` . . . 14
- 3.5. Structure `Circle` . . . 15
- 3.6. Structure `Polygon` . . . 15

4. Structure générique `Shape` **15**
- 4.1. Type structuré `Shape` . . . 16
- 4.2. Gestion des numéros uniques . . . 18

Présentation du projet

Préambule

Une image numérique est un fichier informatique permettant de stocker une image dans la mémoire de l'ordinateur (sous forme binaire). La création de cette image numérique peut être effectuée à partir de la saisie d'une image réelle par l'intermédiaire de dispositifs matériels (appareil photo numérique, caméra numérique, scanner, ...) ou produite complètement par un ordinateur, on parle alors d'image de synthèse. Une fois l'image représentée sous forme binaire, il est possible d'effectuer des traitements (modifications, transformations, filtres, ...) sur l'image grâce à des logiciels graphiques.

Comme souvent en informatique, la manière choisie pour représenter une donnée informatique va conduire à obtenir certains avantages mais aussi des inconvénients en fonction du contexte d'utilisation. Il existe deux types de représentation pour les images numériques :

— `Image matricielle` : les données de l'image sont représentées sous la forme d'une matrice de points à plusieurs dimensions. Pour des images à 2 dimensions les points sont appelés pixels.

Avantages : possibilité de modifier l'image pixel par pixel ce qui permet des nuances de couleurs importantes (dégradés, ombres, ...) et d'avoir des effets de textures.

Inconvénients : fichier lourd (même compressé), perte de qualité lors d'un agrandissement (visuel flou aussi appelé pixélisation de l'image). Une illustration du problème de pixélisation est donné en figure 1.

FIGURE 1 – Problème de pixélisation lors d'un agrandissement d'une zone de l'image matricielle. Image extraite du film *The Matrix* réalisé par Les Wachowski.

- Exemples de formats de fichiers matriciels : jpeg, gif, png, bmp - Exemples de logiciels pour la création d'images matricielles : Photoshop, After Effects, Gimp

— `Image vectorielle` : les données de l'image sont représentées d'un point de vue mathématique. Une image vectorielle décrit uniquement les formes géométriques de l'image avec différents attributs que l'ordinateur est chargé de tracer à l'écran.

Avantages : fichier léger, aucune de perte de qualité lors d'un redimensionnement (pas d'effet de pixélisation). Un exemple d'agrandissement sur une image vectorielle est donné en figure 2.

Inconvénients : Ne permet de représenter que des formes géométriques simples (segments,

arcs, cercles, courbes, ...) donc non adapté à la photographie.

FIGURE 2 – Aucune perte de qualité lors de l'agrandissement d'une zone de l'image

- Exemples de formats de fichiers vectoriels : svg, ai, eps, pdf - Exemples de logiciels pour la création d'images vectorielles : Illustrator, CorelDRAW, *Inkscape*, *librecad*, *Dia Diagram Editor*

Note : Il est possible de convertir les images d'un type vers l'autre.

Dans le cadre de ce projet nous allons nous intéresser à la réalisation des images vectorielles et plus particulièrement au fonctionnement des logiciels de dessin vectoriel. Le dessin vectoriel est très utilisé pour la création d'illustrations, de graphiques ou de cartes, car il offre à l'utilisateur la possibilité d'observer avec précision et sans perte de détails les zones qui l'intéressent en fonction de l'échelle souhaitée.

1. Le dessin vectoriel

1.1. Principe de base

Le principe des logiciels de dessin vectoriel est de stocker les informations minimales permettant de reconstruire une forme géométrique à dessiner.

Par exemple, pour une forme cercle, il suffit de stocker la position du centre et le rayon en mémoire. Ce n'est qu'au moment de l'affichage à l'écran que les positions des pixels à colorier sont calculées afin de faire apparaître la forme géométrique finale. Il est bien évidemment possible d'ajouter d'autres informations en mémoire comme par exemple la couleur du contour du cercle ou sa couleur de remplissage.

Dans les logiciels de dessin vectoriel l'outil plume est très important car il permet de créer des formes plus complexes comme des courbes. Les courbes sont représentées en mémoire par des courbes de Bézier. Ces courbes sont polynomiales dont le degré est variable. Par exemple, une courbe de Bézier cubique est une courbe polynomiale de degré 3 et est définie par 4 points : 2 points correspondant aux extrémités de la courbe et 2 points correspondant à des points de contrôle. La position des points de contrôle permet d'ajuster la courbure.

Pour concevoir des dessins complexes, il est nécessaire de pouvoir travailler sur une partie de l'image sans affecter d'autres parties. L'image globale est décomposée en un ensemble de couches empilées les unes sur les autres. Chaque couche regroupe ainsi une partie des formes géométriques (i.e. les informations permettant sa reconstruction) de l'image suivant certains critères (type de forme, zone de l'image, etc). Il est donc possible de retoucher certaines parties de l'image de manière indépendante. C'est ce qu'on appelle en infographie les **calques**.

Une image vectorielle est composée d'une hiérarchie de plusieurs calques superposés les uns sur les autres dont chacun va regrouper un ensemble de formes géométriques ayant des propriétés communes.

L'ensemble des formes géométriques d'un calque sont elles mêmes hiérarchisées à l'intérieur du calque. Cette hiérarchisation à plusieurs niveaux est très importante car elle précise l'ordre dans lequel les calques, et donc les formes géométriques qui les composent, vont se superposer lors de l'affichage à l'écran. Ainsi, une forme géométrique située à un niveau inférieur dans cette hiérarchie peut être partiellement ou entièrement recouverte par une forme géométrique d'un niveau supérieur (appartenant au même claque ou à un calque de niveau supérieur).

L'utilisation de calques permet ainsi d'appliquer des modifications ciblées sur des groupes de formes géométriques tout en donnant la possibilité de verrouiller l'accès sur le groupe afin d'éviter des erreurs de manipulation. Le système de calques est un élément essentiel pour les logiciels de dessin vectoriel et offre de nombreuses fonctionnalités pour travailler de manière précise sur les éléments d'une image afin de produire des images de qualité.

Ce qu'il faut retenir sur les calques :

- un calque peut contenir plusieurs formes géométriques ;
- une forme géométrique ne peut appartenir qu'à un seul calque ;
- les modifications attribuées sur un calque sont effectuées sur toutes les formes géométriques du calque ;
- les modifications attribuées sur une forme géométrique sont effectuées seulement sur cette forme géométrique ;
- il est possible de déplacer une forme géométrique d'un calque à un autre ;
- si un calque est déplacé dans la hiérarchie des calques, toutes les formes géométriques le constituant sont déplacées avec celui-ci.

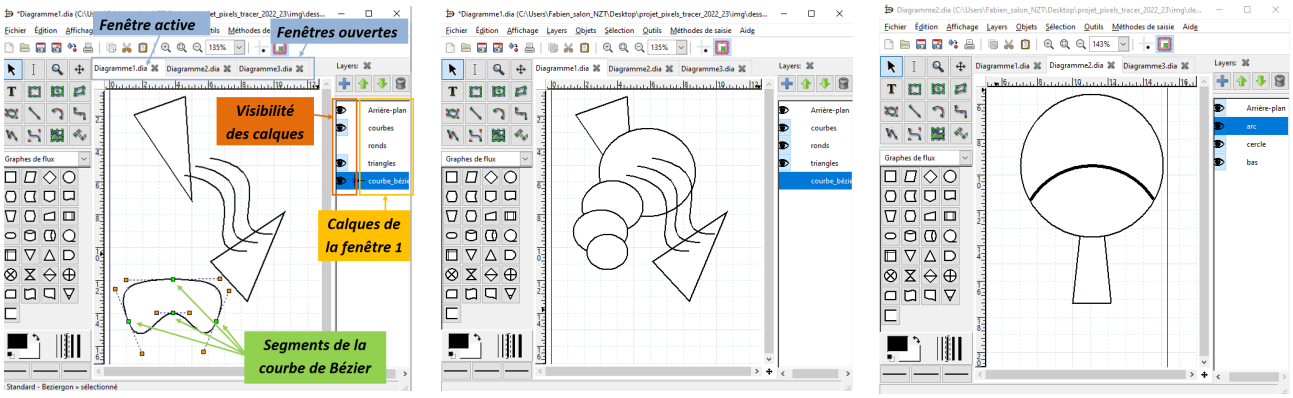
Les opérations principales sur les calques :

- créer un nouveau calque ;
- déplacer un calque dans la hiérarchie (monter ou descendre) ;
- renommer un calque ;
- supprimer un calque : entraîne la suppression des formes géométriques qu'il regroupe ;
- rendre visible ou invisible le calque : permet l'affichage à l'écran ou non des formes géométriques qu'il regroupe ;
- verrouiller ou déverrouiller le calque : autorise la modification ou non des formes géométriques qu'il regroupe.

La section suivante illustre ce système de superposition de calques à partir d'interfaces de logiciels de dessin vectoriel.

1.2. Exemples d'interface de logiciel de dessin vectoriel

Le logiciel Dia Diagram Editor (nommé Dia dans la suite du document) est logiciel de dessin vectoriel orienté pour la création de diagramme. La figure 3 montre l'interface du logiciel Dia. Ce logiciel donne la possibilité à l'utilisateur d'ouvrir plusieurs fenêtres dans une même session pour créer plusieurs images vectorielles et basculer de l'une vers l'autre facilement. Chaque fenêtre est composée d'une superposition de calques regroupant différentes formes géométrique.



(a) Contenu fenêtre 1 avec calque ronds non visible

(b) Contenu fenêtre 1 avec calque courbe de Bézier non visible et calque ronds visible

(c) Contenu fenêtre 2 et ses différents calques

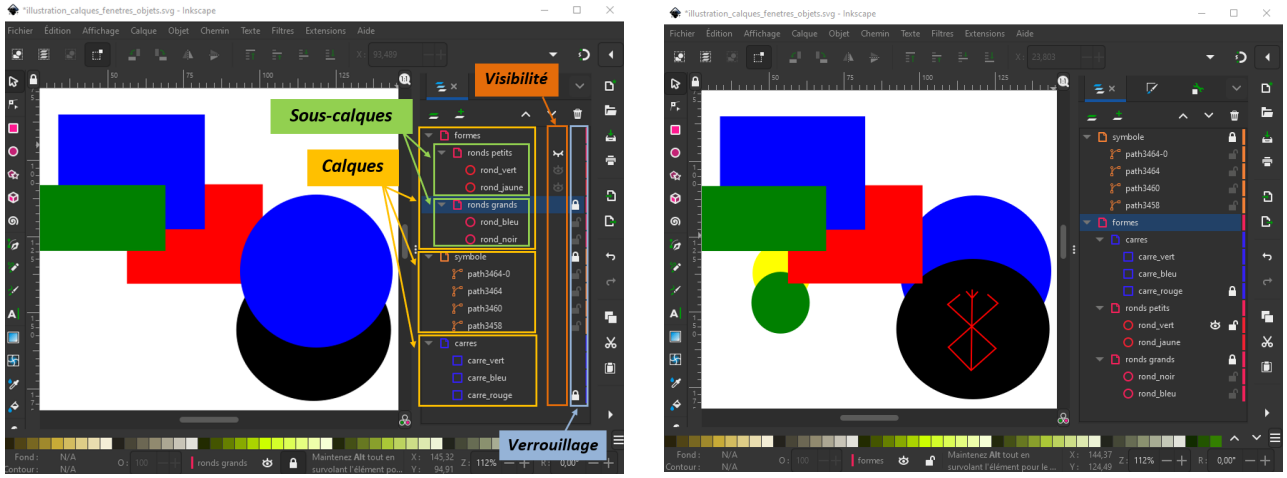
FIGURE 3 – Logiciel Dia Diagram Editor avec son système de calques par fenêtre

Sur la figure 3a le calque `ronds` n'est pas visible donc l'ensemble des formes géométriques de ce calque ne s'affichent pas sur la zone de dessin. On peut observer sur cette même figure une courbe de Bézier assez complexe ses différents points de contrôle visibles.

Sur la figure 3b le calque rond est maintenant visible tandis que le calque `courbe_bézier_fermée` ne l'est plus. Les quatre cercles regroupés dans le calque `ronds` sont donc présents dans la zone de dessin tandis que la courbe de Bézier fermée n'apparaît plus.

Sur la figure 3c c'est la fenêtre 2 qui est active. Dans cette fenêtre tous les calques sont visibles et s'affichent dans la zone de dessin qui lui est propre. Il est intéressant de remarquer le principe de superposition de calques sur cette figure. Dans la hiérarchie de calques de cette fenêtre le calque `ronds` est en dessous du claque `courbes` et au dessus du calque `triangles`. Lors du tracé dans la zone de dessin l'ensemble des quatre cercles du calque `ronds` apparaissent donc par dessus les triangles. Les trois courbes, regroupées dans le calque `courbes`, apparaissent par dessus les triangles et les cercles.

Le logiciel Inkscape est un logiciel de dessin vectoriel gratuit et open source. Il permet de faire des illustrations, des icônes, des logos ou diagrammes. Ce logiciel a les mêmes icônes que le logiciel Dia pour le verrouillage et la visibilité des calques. Il propose cependant la possibilité de créer des calques imbriqués comme illustré sur la figure 4. Un calque peut être composé d'un ensemble de sous-calques, chacun composé d'un ensemble de formes géométriques.



(a) hiérarchie originale

(b) hiérarchie modifiée

FIGURE 4 – Logiciel Inkscape avec son système de calques imbriqués

On peut remarquer sur la figure 4a que le sous-calque `ronds_petits` est non visible. Il n'apparaît donc pas sur le dessin et c'est donc la forme `rond_bleu` qui apparaît au premier plan dans la zone de dessin car il est placé le plus haut dans la hiérarchie et visible.

Sur la figure 4b la hiérarchie de calques a été modifiée par rapport à celle présente sur la figure 4a. Le calque `carres` (de couleur bleu dans zone des calques) a été intégré dans le calque `formes` (de couleur rouge). Il est donc devenu un sous-calque du calque forme et se retrouve au dessus des sous-calques `ronds_petit` et `ronds_grand` (de couleurs rouges). La forme `rond_noir` a été remontée dans la hiérarchie du sous calque `ronds_grands`. Il recouvre donc une partie du rond bleu sur le dessin dorénavant. Le calque `symbole` (de couleur orange) a été déplacé au dessus de tous les autres calques dans la hiérarchie. Ce calque regroupe un ensemble de courbes de Bézier qui apparaissent donc au premier plan sur la zone de dessin (par dessus le rond bleu et le rond noir).

2. Objectifs du projet

Dans ce projet, nous souhaitons développer une application de dessin vectoriel en mode textuel.

L'application à réaliser devra au minimum permettre à un utilisateur la création d'une image vectorielle simple à l'aide d'un ensemble de commandes utilisateurs et d'afficher cette image en mode textuel. L'application doit donc être capable de gérer au minimum une fenêtre et un calque regroupant toutes les formes géométriques créées par l'utilisateur.

Pour ce faire, nous devons nous intéresser à trois aspects principaux :

— la représentation mémoire de toutes les informations de notre application. En particulier la manière de sauvegarder les informations des formes géométriques permettant de les afficher ;

— une interface en ligne de commandes (sous la forme d'un menu) permettant à l'utilisateur d'effectuer les actions permises par l'application ;

— l'affichage à l'écran en mode textuel de l'image créée en appliquant les algorithmes de dessin décrits dans les sections correspondantes.

A partir de cette version minimale, il vous sera demandé d'implémenter diverses améliorations comme par exemple :

— la création de forme plus complexes : les courbes de Bézier ;
— une optimisation du code pour l'ajout, la suppression ou le déplacement d'une forme dans la hiérarchie (utilisation de listes chaînées) ;
— un système de calques multiples (imbriqués ou non) ;
— un système de fenêtres multiples ;

Vous pourrez bien évidemment implémenter d'autres améliorations comme la possibilité : d'appliquer des transformations sur les formes de l'image (rotation, translation, etc), zoomer/dézoomer, de déplacer une forme d'un calque à un autre, déplacer des calques dans la, de sauvegarder / charger une image dans un fichier texte, de proposer des commandes permettant d'annuler ou refaire une ancienne commande (Undo / Redo).

2.1. Représentation mémoire des données

L'idée est de réfléchir à la représentation des formes géométriques, des fenêtres et des calques. La figure 5 illustre une possible organisation des formes. Sur cette figure on peut constater que les données dans l'application sont stockés comme des tableaux imbriqués. L'application possède plusieurs fenêtres (tableau de fenêtres) dont chacune est composée d'un tableau de calques. Chaque calque peut contenir un ensemble de formes géométriques identiques ou différentes.

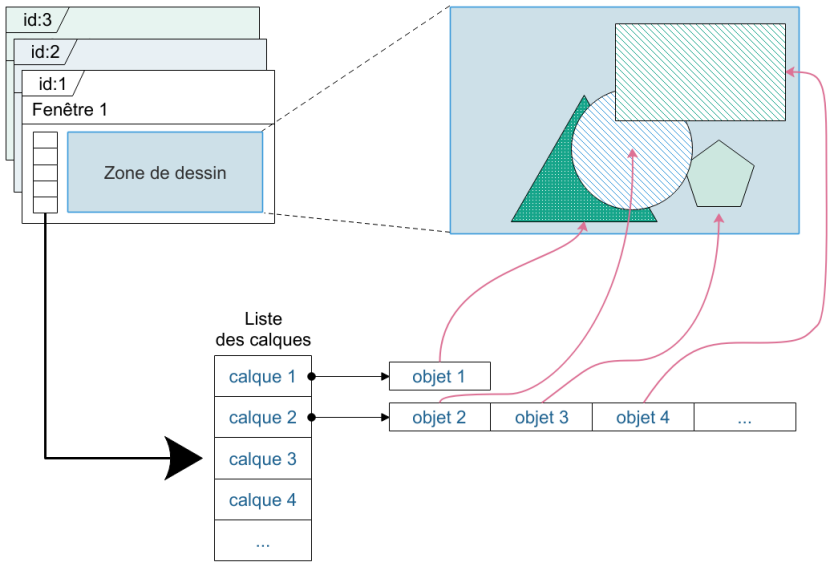


FIGURE 5 – Schéma mémoire simplifié d'une application de dessin vectoriel.

2.2. Interface en ligne de commande

Nous allons devoir créer un menu permettant à l'utilisateur d'effectuer les actions permises dans le logiciel telles que l'ajout d'une forme géométrique au calque ou l'affichage des informations des formes déjà ajoutées.

Cette interface utilisateur se présentera sous la forme d'un menu pour l'utilisateur.

Exemple de menu :

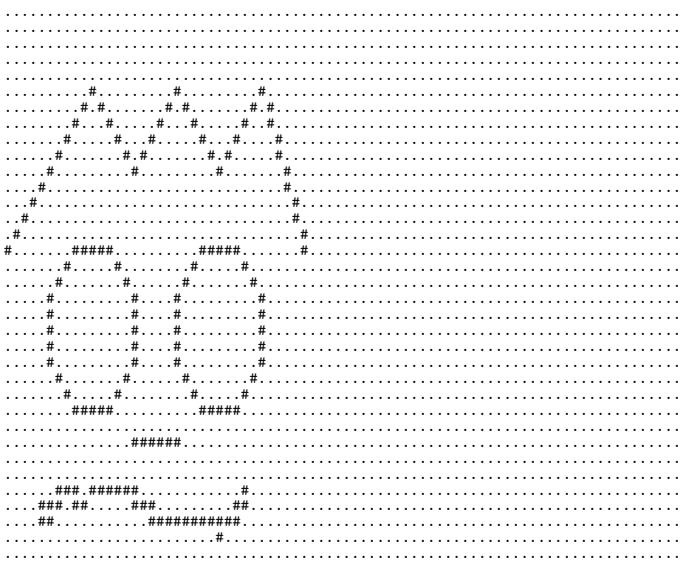
```
Veuillez choisir une action :
    A- Ajouter une forme
    B- Afficher la liste des formes
    C- Supprimer une forme
    D- Tracer le dessin
    E- Aide
    [Autres actions]
>> Votre choix : A
    Veuillez choisir une action :
        1- Ajouter un point
        2- Ajouter une ligne
        3- Ajouter cercle
        4- Ajouter un carre
        5- Ajouter un rectangle
        6- Ajouter un polygone
        7- Revenir au menu precedent
    >> Votre choix : 2
        Saisir les informations de la ligne :
        >> Saisir le premier point x1 y1 : 12 16
        >> Saisir le deuxieme point x2 y2 : 14 18
    >> Votre choix : 3
        Saisir les informations du cercle :
        >> Saisir le point centre x1 y1 : 24 10
        >> Saisir le rayon du cercle : 7
    >> Votre choix : 7
Veuillez choisir une action :
    A- Ajouter une forme
    B- Afficher la liste des formes
    C- Supprimer une forme
    D- Tracer le dessin
    E- Aide
    [Autres actions]
>> Votre choix : B
    Liste des formes :
        1 : CIRCLE 20 10 5
        2 : CIRCLE 20 25 5
        3 : LINE 5 20 10 25
        4 : POLYGON 15 0 5 10 10 15 5 20 10 25 5 30 15 35
        5 : CURVE 35 5 25 5 40 30 30 30
>> Votre choix : D
```

2.3. Génération du dessin à l'écran

Pour dessiner les différentes formes géométriques à l'écran, il existe des algorithmes de tracé pour chaque forme de base. Ceux-ci permettent de générer une approximation de la forme désirée sur la base de ses paramètres. Le détail de chacun de ces algorithmes est décrit dans la partie 2.

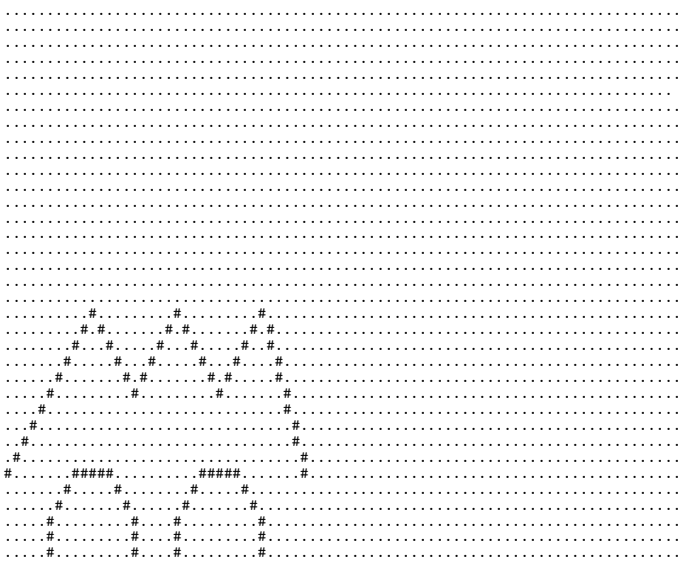
Exemple de dessin généré :

```
Veuillez choisir une action :
    A- Ajouter une forme
    B- Afficher la liste des formes
    C- Supprimer une forme
    D- Tracer le dessin
    E- Aide
    [Autres actions]
>> Votre choix : B
    Liste des formes :
        1 : CIRCLE 20 10 5
        2 : CIRCLE 20 25 5
        3 : LINE 5 20 10 25
        4 : POLYGON 15 0 5 10 10 15 5 20 10 25 5 30 15 35
        5 : CURVE 35 5 25 5 40 30 30 30
>> Votre choix : D
```



Il est à noter que les formes géométriques créées à partir de coordonnées se retrouvant partiellement ou totalement à l'extérieur de la zone de dessin sont acceptées. Celles-ci seront stockées en mémoire mais n'apparaiteront pas à l'écran.
Dans le cas où seulement une partie de la forme est à l'extérieur de la zone de dessin, il est nécessaire d'afficher à l'écran la portion du dessin se retrouvant à l'intérieur de la zone de dessin.

Exemple : Même figure dessinée avec des coordonnées différentes



Première partie
Structures de données des formes

Pour débuter ce travail, nous allons d'abord développer la partie du code de l'application permettant de créer et de stocker des formes spécifiques : point, segment, cercle, carré, rectangle et polygone. On ne s'intéressera donc pas à la notion de calques, de fenêtres ou du dessin des formes à l'écran pour le moment. Comme nous le verrons plus en détails à la section 4, nous allons devoir également créer un type abstrait (générique) pour la gestion de ces formes. Ce type générique sera indépendant d'une forme particulière afin de faciliter les développements qui suivront dans la deuxième partie.

Dans cette partie nous allons créer un module composé des fichiers `shapes.c` et `shapes.h`. Ce module va regrouper les types structurés de chaque forme ainsi que les fonctions associées : de création, d'affichage et de destruction. On fait référence dans cette partie aux fonctions d'affichage permettant d'afficher les informations liées à chacune des formes et non du tracé de ces formes à l'écran.

3. Les structures spécifiques des formes disponibles

3.1. Structure `Point`

Un point est une forme graphique qui représente une position dans l'espace. Notre application de dessin vectoriel étant textuelle nous utiliserons un caractère de remplissage pour le représenter dans notre espace à 2 dimensions. Dans la suite du document, nous prendrons comme caractère de remplissage le '#'.

La représentation en mémoire d'un point se fera par une variable de type structuré `Point` composé de deux champs entiers `pos_x` et `pos_y` représentant ses coordonnées sur les axes x et y.

```
typedef struct {
    int pos_x;
    int pos_y;
}Point;
```

Pour manipuler un point, les fonctions suivantes doivent être définies :

```
Point *create_point(int px, int py);
void delete_point(Point * point);
void print_point(Point * p);
```

Où :

- La fonction `Point *create_point(int px, int py)` permet d'allouer dynamiquement une variable de type structuré `Point` dont les coordonnées sont données en paramètre.
- La fonction `void delete_point(Point * point)` permet de libérer la mémoire allouée au point donné en paramètre.
- La fonction `void print_point(Point * p)` permet d'afficher à l'écran les informations d'un `Point` sous la forme suivante : `POINT [p(pos_x, pos_y)]`

Exemple d'utilisation

```
Point * p = create_point (10, 15);
print_point(p);
delete_point(p);
```

Résultat à l'écran

```
POINT 10 15
```

3.2. Structure `Line`

Une ligne est une forme géométrique correspondant à un segment délimité par deux points. Elle est représentée en mémoire comme une structure composée de deux champs de type `Point` représentant les extrémités du segment. Les variables de type `Point` étant créées dynamiquement, les champs de la structure devront être de type `Point*`.

```
typedef struct line {
    Point *p1;
    Point *p2;
}Line;
```

Les fonctions associées à ce type structuré sont les suivantes :

```
Line *create_line(Point * p1, Point * p2);
void delete_line(Line * line);
void print_line(Line * line);
```

Où :

- La fonction `Line *create_line(Point * p1, Point * p2)` permet d'allouer dynamiquement un segment de type structuré `Line` à partir de deux points donnés en paramètres.
- La fonction `void delete_line(Line * line)` permet de libérer la mémoire allouée au segment donné en paramètre.
- `void print_line(Line * line)` permet d'afficher les informations d'un segment selon le format décrit dans l'exemple ci-après : `LINE [x1, y1, x2, y2]`

Exemple d'utilisation

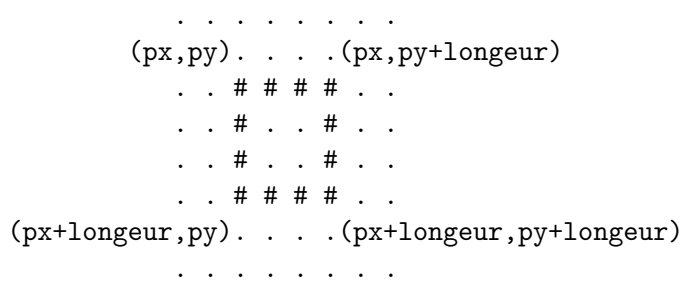
```
Point * p1 = create_point (10, 15);
Point * p2 = create_point (21, 25);
Line * l = create_line (p1 ,p2);
print_line (l);
delete_line(l);
delete_point(p1);
delete_point(p2);
```

Résultat à l'écran

```
LINE 10 15 21 25
```

3.3. Structure `Square`

La forme carré est représentée par un point dans l'espace (coin supérieur gauche) et une longueur. À partir de ces informations vous définirez vous même la structure `Square`. Les positions des 3 autres coins peuvent être calculées comme suit :

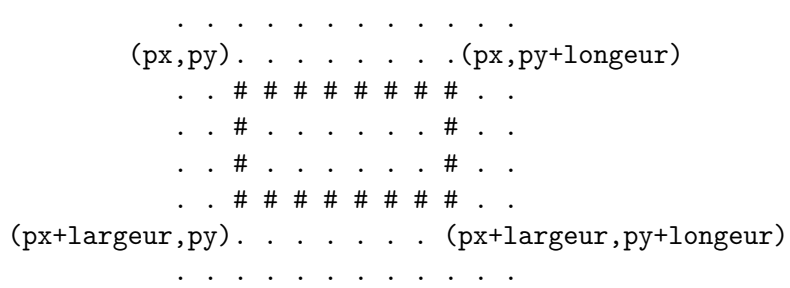


Un carré est donc représenté en mémoire comme une structure ayant deux champs : un point et une longueur. De la même façon que pour les structures précédentes, il est nécessaire d'implémenter trois fonctions associées à la manipulation de ce type structuré :

```
Square *create_square(Point * point, int length);
void delete_square(Square * square);
void print_square(Square * square);
```

3.4. Structure `Rectangle`

La forme rectangle est représentée par un point (coin supérieur gauche), une longueur et une largeur. À partir de ces informations vous définirez vous même la structure `Rectangle`. Les positions des 3 autres coins peuvent être calculées comme suit :



Tout comme pour les structures précédentes, il est nécessaire d'implémenter trois fonctions pour la manipulation de ce type structuré :

```
Rectangle *create_rectangle(Point * point, int width, int height);
void delete_rectangle(Rectangle * rectangle);
void print_rectangle(Rectangle * rectangle);
```

3.5. Structure `Circle`

La forme cercle est représentée par un point et un rayon. À partir de ces informations vous définirez vous même la structure `Circle`. De la même façon que pour les structures précédentes, il est nécessaire d'implémenter trois fonctions associées à la manipulation de ce type structuré :

```
Circle *create_circle(Point * center, int radius);
void delete_circle(Circle * circle);
void print_circle(Circle * circle);
```

3.6. Structure `Polygon`

Le polygone est représenté comme un ensemble de points à relier. Le type structuré de cette forme est constitué de deux champs : un tableau dynamique de points et sa taille `n`. Le tableau de points doit être dynamique (taille logique égale à la taille physique) car la taille `n` n'est pas connue à la compilation, c'est l'utilisateur qui le précisera avec la commande de création d'un polygone.

Note : Il est important de rappeler que dans ce projet les variables de type *Point* sont créés dynamiquement par la fonction `create_point` qui retourne donc une variable de type *Point**. Le deuxième champ de la structure `Polygon` doit être déclaré comme un tableau 1D dynamique dont chaque case est de type `Point*`. Ce champ est donc de type *Point***.

NB : Pour avoir un polygone fermé, il est nécessaire de que les coordonnées de son premier point soient les mêmes que celles du dernier point.

La structure d'un polygone est la suivante :

```
typedef struct polygon {
    int n;
    Point ** points; // tableau 1D dynamique de Point*
}Polygon;
```

Les fonctions qui lui sont associées sont encore une fois similaires à ce qui a été fait pour les précédentes structures :

```
Polygon *create_polygon(int n);
void delete_polygon(Polygon * polygon);
void print_polygon(Polygon * polygon);
```

4. Structure générique `Shape`

Nous avons maintenant à disposition un ensemble de fonctions spécifiques pour gérer chacune de nos formes. Cependant le système de calque présenté dans la partie 1.1 à la page 5 doit permettre de regrouper des formes de type différent. De plus, chaque forme doit pouvoir être identifiée de manière unique dans l'application. Ce numéro doit être indépendant de la forme mais aussi du calque auquel la forme appartient car l'utilisateur doit avoir la possibilité de faire basculer une forme d'un calque à un autre.

L'application doit donc être en mesure de manipuler un type abstrait permettant de représenter un type de forme quelconque : *Point*, *Line*, *Square*, etc. C'est ce qu'on appelle la **généricité** en programmation. En langage C cette généricité est obtenue avec un type particulier de pointeur : `void *`.

Compléments d'information sur le type `void*`

Pour rappel, un pointeur est une variable permettant de stocker l'adresse d'une autre variable du programme. Le pointeur pointe ou référence une autre variable. L'opérateur de déréférencement "*" appliqué sur un pointeur permet de récupérer la valeur de la variable référencée. Cet opérateur de déréférencement ne fonctionne que si le type de la variable référencée est connue : c'est pour cette raison qu'il est obligatoire de préciser le type de la variable référencée lors de la déclaration d'un pointeur.

Le type `void*` est un type de pointeur dit **générique** : il permet de stocker l'adresse d'une variable dont le type n'est pas connue. **L'opérateur de déréférencement sur ce type de pointeur est illégale** car la machine ne sait pas combien d'octets lire à cette adresse pour récupérer la valeur, ni la convention de codage binaire utilisée par ailleurs (cf. module de système numérique !).

Vous avez déjà utilisé des fonctions génériques utilisant le type `void*` : `printf` et `malloc`.

La fonction `malloc` permet d'allouer dynamiquement n'importe quel type de variable ou de tableau. Il n'existe qu'une seule version de la fonction `malloc` quelque soit le type de ce qui est alloué (variable de type `int`, tableau 1D de `char`, tableau 2D de `float`, etc). La généricité de cette fonction est obtenue par le type de la valeur de retour qui est `void*`. La fonction renvoie ainsi l'adresse d'une zone mémoire de n'importe quel type. Comme le savez, cette valeur de retour doit être **castée** (implicitement ou explicitement par le programmeur) vers un nouveau pointeur du type attendu pour être utilisée dans la suite du code.

La fonction `printf` permet avec le format `%p` d'afficher n'importe quel type d'adresse. Dans ce cas de figure, la généricité de la fonction est obtenue en utilisant un paramètre de type `void*`. Lors de l'appel à la fonction, le pointeur ou l'adresse spécifiée en argument est `casté` implicitement vers un `void*` afin d'être afficher.

Ce qu'il faut retenir sur les pointeurs de type `void*` :

- ce type de pointeur permet de pointer sur n'importe quel type de variable (types de base, structurés, etc) ;
- ce type est utilisé pour rendre des fonctions génériques du point de vue de leur entrées et/ou sortie, c'est à dire indépendante d'un type particulier.
- L'arithmétique des pointeurs n'existe pas sur ce type de pointeur. Il est obligatoire de `caster` le pointeur (implicitement ou explicitement) vers le type souhaité pour qu'une opération de déréférencement soit possible par exemple.
- Attention à l'ambiguïté du mot clé `void` l'expression `void*`. Un pointeur de ce type ne pointe pas vers une variable de type `void` et cette expression ne doit pas non plus être interprétée comme un pointeur vers "rien".

4.1. Type structuré `Shape`

Pour gérer cette généricité pour les formes nous allons créer un type structuré `Shape` utilisant le type particulier `void*` pour le champ permettant de pointer vers n'importe quel type de structure correspond aux formes que nous avons définie dans la section précédente. Un champ supplémentaire représentera le numéro d'identification (nommé `ID` dans la suite du document) afin d'identifier de manière unique la forme dans l'application.

Note : Ce type générique donne la possibilité d'ajouter une nouvelle forme à notre application en limitant l'impact sur le code déjà écrit. Cette type d'avantage correspond à l'indicateur appelé maintenabilité pour la qualité logiciel.

Ainsi, notre nouveau type structuré Shape sera composé d'un identifiant (id), un pointeur générique vers n'importe quelle forme ainsi que le type de la forme pointée (`SHAPE_TYPE shape_type`).

Étant donné qu'il existe une liste limitée de formes géométriques, nous allons utiliser une énumération pour leurs types.

```
typedef enum { POINT, LINE, SQUARE, RECTANGLE, CIRCLE, POLYGON} SHAPE_TYPE;

typedef struct shape {
    int id;                  // identifiant unique de la forme
    SHAPE_TYPE shape_type;   // type de la forme pointé
    void *ptrShape;          // pointeur sur n'importe quelle forme
}Shape;
```

Afin de manipuler ces formes de façon générique, nous devons implémenter les fonctions suivantes :

```
Shape *create_empty_shape(SHAPE_TYPE shape_type);
Shape *create_point_shape(int px, int py);
Shape *create_line_shape(int px1, int py1, int px2, int py2);
Shape *create_square_shape(int px, int py, int length);
Shape *create_rectangle_shape(int px, int py, int width, int height);
Shape *create_circle_shape(int px, int py, int radus);
Shape *create_polygon_shape(int lst[], int n);
void delete_shape(Shape * shape);
void print_shape(Shape * shape);
```

L'idée est que la fonction `Shape *create_empty_shape(SHAPE_TYPE shape_type)` alloue la zone mémoire qui contiendra un type de forme donné en paramètre. Pour ce faire, nous proposons de l'écrire comme suit :

```
Shape *create_empty_shape(SHAPE_TYPE shape_type) {
    Shape *shp = (Shape *) malloc(sizeof(Shape));
    shp->ptrShape = NULL;
    shp->id = 1; // plus tard on appelera get_next_id();
    shp->shape_type = shape_type;
    return shp;
}
```

Puis, créer une fonction générique pour chaque type de forme. Nous vous proposons le code de la fonction `Shape *create_empty_shape(SHAPE_TYPE shape_type)` ci-dessous :

```
Shape *create_point_shape(int px, int py) {
    Shape *shp    = create_empty_shape(POINT);
    Point *p      = create_point(px, py);
    shp->ptrShape = p;
    return shp;
}
```

Il est donc demandé de comprendre et recopier les deux fonctions fournies puis de déduire le code des fonctions suivantes :

— `Shape *create_line_shape(int px1, int py1, int px2, int py2)` permet de de créer une forme `Shape` vide, puis lui associe une forme de type ligne ;

— La fonction `Shape *create_square_shape(int px, int py, int length)` permet de créer une formr `Shape` vide, puis lui associe une forme de type carré ;

— La fonction `Shape *create_rectangle_shape(int px, int py, int width, int height)` permet de créer une forme `Shape` vide, puis lui associe une forme de type rectangle ;

— La fonction `Shape *create_cercle_shape(int px, int py, int radus)` permet de créer une forme `Shape` vide, puis lui associe une forme de type cercle

— La fonction `Shape *create_polygon_shape(int lst[], int n)` permet de créer une forme `Shape` vide, puis lui associe une forme de type polygone. Cette fonction doit aussi vérifier que le nombre de points n est un multiple de deux, car le tableau doit contenir un nombre pair d'éléments ;

— La fonction `void delete_shape(Shape * shape)` permet de supprimer une forme.

— La fonction `void print_shape(Shape * shape)` permet d'afficher à l'écran l'id de la *shape* son type puis appellera la fonction `print_...` de la forme encapsulée (voir l'exemple ci-après).

Exemple d'utilisation

```
Shape f1 = create_line_shape (10, 15, 21, 25);
print_shape (f1);
delete_shape (f1);
```

Résultat à l'écran

```
LINE 10 15 21 25
```

4.2. Gestion des numéros uniques

Les formes (*Shapes*) sont identifiées par un numéro unique. Pour garantir cette unicité, la méthode à utiliser est la suivante :

1. Créer une variable globale **global_id** initialisée à 0
2. Incrémenter la valeur de cette variable à chaque création d'une forme géométrique. Elle est donc modifiée dans toutes les fonctions de création des formes quelques soient leurs types.

Nous vous proposons de le faire comme suit dans les fichiers `id.h` et `id.c`.

```
unsigned int global_id = 0;
unsigned int get_next_id();
```

— La fonction `unsigned int get_next_id()` permet d'incrémenter le compteur et de retourner sa dernière valeur. Notez que la première forme créée aura pour **id** la valeur 1.

— Le **23/04/2023 à 23h59**, toutes les équipes doivent être capables de déposer sur Moodle une archive **.zip** contenant les fonctionnalités A et B du menu montré dans la section 2.2 à savoir : :

1. L'ajout des formes géométriques à la structure de stockage en mémoire sur la base des structures et prototypes de fonctions proposés ;
2. L'affichage textuel (sans le dessin) des formes géométriques ainsi que de la liste des formes stockées.

— L'archive à déposer doit contenir :

— Des fichiers **.c** et des fichiers **.h**

— Un mini rapport expliquant le squelette global de la première partie du projet : Schéma de la structure utilisée pour le stockage en mémoire ainsi que l'ordre d'appel des fonctions (structure du menu au format texte ou organigramme)

Si vous voyez ce message, c'est que vous avez bien tout lu, Bravo ! Avancez bien sur cette partie pour avoir la prochaine ☺