C Programming Project

Vector Text-based Editor

Halim Djerroud, Fabien Calcado, Asma Gabis

March 2023

Instructions & General Information :

➔ This project is to be done exclusively in C language

➔ Project Attachment :

- An executable (Windows version) to give you an idea of how it works and some of the features you want

➔ Teams organization :

- This project is to be carried out in pairs (A single group of three students could be accepted only in case the number of students of a group is odd)
- The list of teams is to be given to the teachers at the **latest** at the end of the first project follow-up session

➔ Project Filing : Two filings are required:

- An intermediate deposit for part 1 of the project
- A final submission of the entire project before the defense

➔ Key dates:

- Publication date part 1 : **03/18/2023**
- Follow-up session date 1: week of **03/20/2023**
- Publication date part 2 : **03/XX/2023**
- Follow-up session date 2: week of **04/17/2023**
- Intermediate filing : **04/23/2023 à 11h59 pm**
- Follow-up session date 3: week of **05/15/2023**
- Final deposit date : **05/21/2023 à 11h59 pm**
- Date of defense: Week of **05/22/2023**

➔ Final deposit : This is to be done on Moodle as a **.zip** archive containing:

- The project code with all **.h** and **.c** files
- The report in **.pdf**
- A **README.txt** file listing the programs and how to use them in practice. This file should contain all the necessary instructions to explain to the user how to compile and use your software.

➔ Evaluation

- Detailed scale: to be provided later
- Final project grade = Code grade + Report grade + Defense grade
- **Reminder :** project grade = 15% of the grade for the course "Algorithms and Data Structures 1"
- Members of the same team may receive different grades depending on the effort put into the project.

Contents

Presentation of the project

Preamble

A digital image is a computer file allowing to store an image in the memory of the computer (in binary form). The creation of this digital image can be carried out starting from the seizure of a real image by the intermediary of material devices (numerical camera, numerical camera, scanner, ...) or produced completely by a computer, one speaks then about image of synthesis. Once the image represented in binary form, it is possible to carry out treatments (modifications, transformations, filters, ...) on the image thanks to graphic software.

As is often the case in computer science, the way chosen to represent a computer data will lead to advantages but also to some disadvantages depending on the context of use. There are two types of representation for digital images:

- `Raster image` : the image data is represented as a matrix of points in several dimensions. For 2-dimensional images the points are called pixels.

  Advantages : possibility to modify the image pixel by pixel which allows important color nuances (gradients, shadows, ...) and to have textures effects.

  Disadvantages : heavy file (even compressed), loss of quality when enlarged (blurred visual also called pixelation of the image). An illustration of the pixelation problem is given in figure 1.

Figure 1: Pixelation problem when enlarging an area of the raster image. Image extracted from the movie *The Matrix* directed by Les Wachowski.

- Examples of raster file formats: jpeg, gif, png, bmp - Examples of software for creating raster images: Photoshop, After Effects, Gimp

- `Vector image` : the image data is represented from a mathematical point of view. A vector image describes only the geometric shapes of the image with different attributes that the computer is responsible for drawing on the screen.

  Advantages : light file, no loss of quality when resizing (no pixelation effect). An example of enlargement on a vector image is given in figure 2.

  Disadvantages : Only allows to represent simple geometrical shapes (segments, arcs, circles, curves, ...) so not suitable for photography.

Figure 2: No loss of quality when enlarging an area of the image

- Examples of vector file formats: svg, ai, eps, pdf
- Examples of software for creating vector images: : Illustrator, CorelDRAW, *Inkscape*, *librecad*, *Dia Diagram Editor*

Note: It is possible to convert images from one type to another.

In this project we are going to look at the creation of vector images and more specifically at the functioning of vector drawing software. Vector drawing is widely used for the creation of illustrations, graphics or maps, because it offers the user the possibility of observing with precision and without loss of detail the areas that interest him according to the desired scale.

1. Vector drawing

1.1. Basic principle

The principle of vector drawing software is to store the minimum information needed to reconstruct a geometric shape to be drawn.

For example, for a circle shape, it is sufficient to store the position of the center and the radius in memory. It is only at the time of display on the screen that the positions of the pixels to be colored are calculated in order to make the final geometric shape appear. It is of course possible to add other information in memory, such as the color of the circle's outline or its filling color.

In vector drawing software the pen tool is very important because it allows you to create more complex shapes like curves. Curves are represented in memory by Bézier curves. These curves are polynomial with variable degrees. For example, a cubic Bézier curve is a polynomial curve of degree 3 and is defined by 4 points: 2 points corresponding to the ends of the curve and 2 points corresponding to control points. The position of the control points allows to adjust the curvature.

To design complex drawings, it is necessary to be able to work on one part of the image without affecting other parts. The global image is decomposed into a set of layers stacked on top of each other. Each layer gathers a part of the geometric shapes (i.e. the information allowing its reconstruction) of the image according to certain criteria (type of shape, image area, etc.). It is therefore possible to retouch certain parts of the image independently. This is what we call **layers** in computer graphics.

A vector image is composed of a hierarchy of several layers superimposed on each other, each of which groups together a set of geometric shapes with common properties. All the geometric shapes of a layer are themselves hierarchical within the layer. This multi-level hierarchy is very important because it specifies the order in which the layers, and therefore the geometric shapes that make them up, will be superimposed when displayed o n t h e screen. Thus, a geometric shape located at a lower level in this hierarchy can be partially or entirely covered by a geometric shape of a higher level (belonging to the same layer or to a higher level layer).

The use of layers makes it possible to apply targeted modifications to groups of geometric shapes while giving the possibility of locking access to the group in order to avoid manipulation errors. The layer system is an essential element for vector drawing software and offers many features for working precisely on the elements of an image in order to produce quality images.

Things to remember about layers :

- a layer can contain several geometric shapes;
- a geometric shape can only belong to one layer;
- the modifications attributed on a layer are carried out on all the geometrical shapes of the layer;
- the modifications attributed on a geometrical shape are carried out only on this geometrical shape;
- it is possible to move a geometric shape from one layer to another;
- if a layer is moved in the layer hierarchy, all the geometric shapes that make up the layer are moved with it.

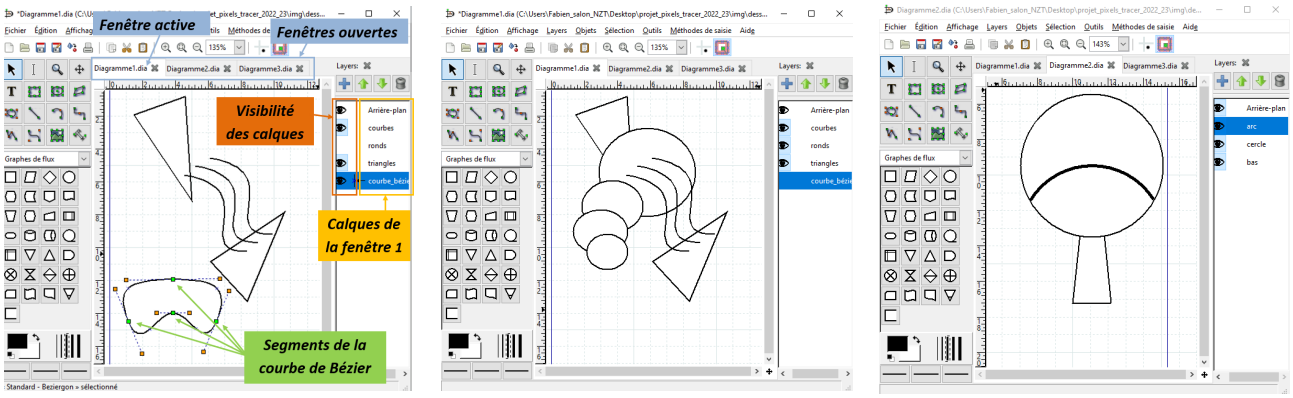
The main operations on layers :

- create a new layer ;
- move a layer in the hierarchy (up or down) ;
- rename a layer ;
- delete a layer: deletes the geometric shapes it contains;
- make the layer visible or invisible: allows the display on the screen or not of the geometrical shapes it contains;
- lock or unlock the layer: authorizes the modification or not of the geometric shapes it contains.

The following section illustrates this layer overlay system from vector drawing software interfaces.

1.2. Examples of vector drawing software interface

The Dia Diagram Editor software (named Dia in the following) is a vector drawing software for creating diagrams. Figure 3 shows the interface of the Dia software. This software gives the user the possibility to open several windows in the same session to create several vector images and to switch from one to the other easily. Each window is composed of an overlay of layers regrouping different geometric shapes.



(a) Contents window 1 with round layer not visible

(b) Contents of window 1 with Bézier curve layer not visible and round layer visible

(c) Content window 2 and its different layers

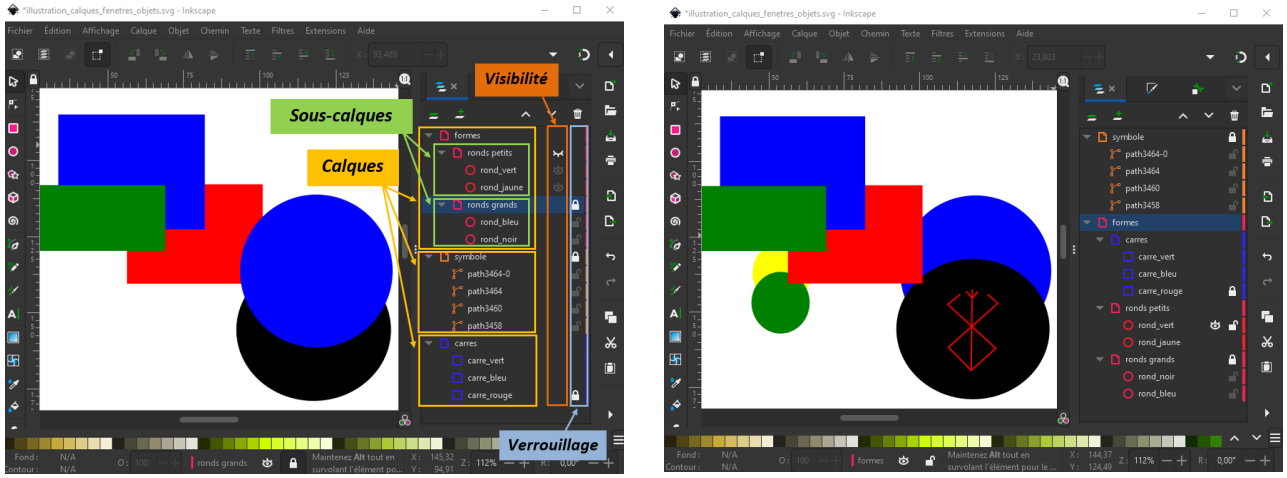
Figure 3: Dia Diagram Editor software with its windowed layer system

On figure 3a the `rond` layer is not visible so all the geometric shapes of this layer are not displayed on the drawing area. On this same figure we can see a rather complex Bézier curve with its various visible control points.

In figure 3b the *rond* layer r is now visible while the closed `courbe_bézier_fermée` layer is not. The four circles grouped in the `ronds` layer are therefore present in the drawing area while the closed Bézier curve is no longer visible.

In figure 3c the window 2 is active. In this window all layers are visible and are displayed in their own drawing area. It is interesting to note the principle of overlapping layers in this figure. In the hierarchy of layers in this window the `ronds` layer is below the `courbes` layer and above the `triangles` layer. When drawing in the drawing area, all four circles on the `ronds` layer appear on top of the triangles. The three curves, grouped in the `courbes` layer, appear on top of the triangles and circles.

Inkscape is a free and open source vector drawing software. It allows you to make illustrations, icons, logos or diagrams. This software has the same icons as the Dia software for locking and visibility of layers. However, it offers the possibility of creating nested layers as shown in figure 4. A layer can be composed of a set of sublayers, each composed of a set of geometric shapes.



(a) original hierarchy

(b) modified hierarchy

Figure 4: Inkscape software with its nested layer system

It can be seen in Figure 4a that the sub-layer `ronds_petits` is not visible. It does not appear on the drawing and it is therefore the `rond_bleu` shape that appears in the foreground in the drawing area because it is placed highest in the hierarchy and visible.

In figure 4b the layer hierarchy has been modified from that shown in figure 4a. The layer "text" (blue in the layers area) has been integrated into the layer "shapes" (red). It has therefore become a sub-layer of the shape layer and is found above the sub-layers "small" and "large" (red colors). The shape `rond_noir` has been moved up in the hierarchy of the sub-layer `ronds_grands`. It therefore covers part of the blue circle on the drawing from now on. The layer `symbol` (of orange color) was moved above all the other layers in the hierarchy. This layer contains a set of Bézier curves which appear in the foreground of the drawing area (on top of the blue circle and the black circle).

2. Objectives of the project

In this project, we want to develop a vector drawing application in text mode.

The application to be realized must at least allow a user to create a simple vector image using a set of user commands and to display this image in textual mode. The application must therefore be able to manage at least one window and one layer containing all the geometric shapes created by the user.

To do this, we need to look at three main aspects:

- the memory representation of all the information of our application. In particular the way to save the information of the geometrical shapes allowing to display them;
- A command line interface (As a menu) allowing the user to perform the actions permitted by the application;
- the display on the screen in text mode of the image created by applying the drawing algorithms described in the corresponding sections.

From this minimal version, you will be asked to implement various improvements such as :

- the creation of more complex shapes: Bézier curves;

- code optimization for adding, deleting or moving a shape in the hierarchy (use of linked lists);
- a system of multiple layers (nested or not) ;
- a multiple window system;

You will of course be able to implement other improvements such as the ability to: apply transformations on the shapes of the image (rotation, translation, etc.), zoom in/out, move a shape from one layer to another, move layers in the image, save/load an image in a text file, offer commands to undo or redo an old command (Undo / Redo).

2.1. Memory representation of data

The idea is to think about the representation of geometric shapes, windows and layers. Figure 5 illustrates a possible organization of shapes. On this figure we can see that the data in the application are stored as nested arrays. The application has several windows (window array), each of which is composed of an array of layers. Each layer can contain a set of identical or different geometric shapes.

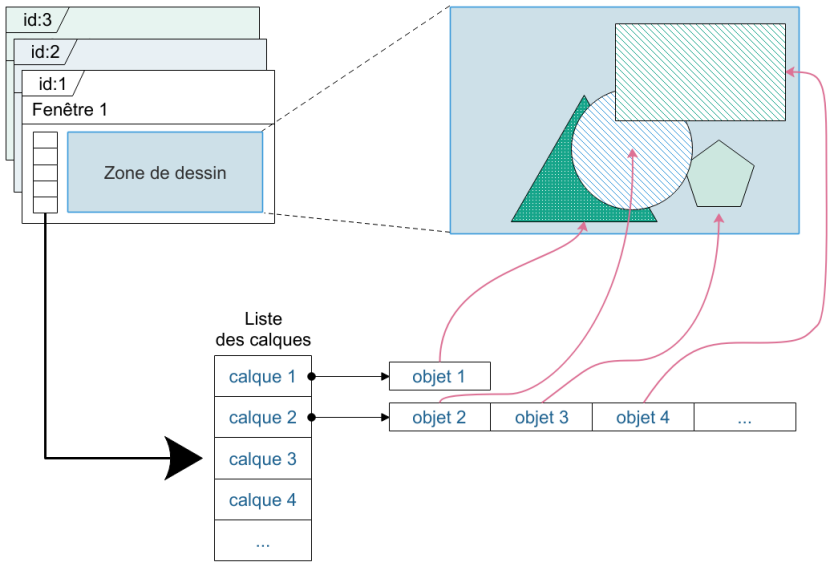


Figure 5: Simplified memory scheme of a vector drawing application.

2.2. Command line interface

We will have to create a menu allowing the user to perform the actions allowed in the software such as adding a geometric shape to the layer or displaying the information of the shapes already added.

This user interface will be presented as a menu for the user

Example of a menu :

```
Please select an action:
    A- Add a shape
    B- Display the list of shapes
    C- Delete a shape
    D- Drawing the shapes
    E- Help
    [Other actions]
>> Your choice: A
    Please select an action:
        1- Add a point
        2- Add a line
        3- Add circle
        4- Add a square
        5- Add a rectangle
        6- Add a polygon
        7- Return to previous menu
    >> Your choice: 2
        Enter the information of the line :
        >> Enter the first point x1 y1 : 12 16
        >> Enter the second point x2 y2 : 14 18
    >> Your choice: 3
        Enter the circle information:
        >> Enter the center point x1 y1 : 24 10
        >> Enter the radius of the circle: 7
    >> Your choice: 7
Please select one action:
    A- Add a shape
    B- Display the list of shapes
    C- Delete a shape
    D- Drawing the shapes
    E- Help
    [Other actions]
>> Your choice: B
    List of shapes :
        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
>> Your choice: D
```

2.3. Drawing on the screen

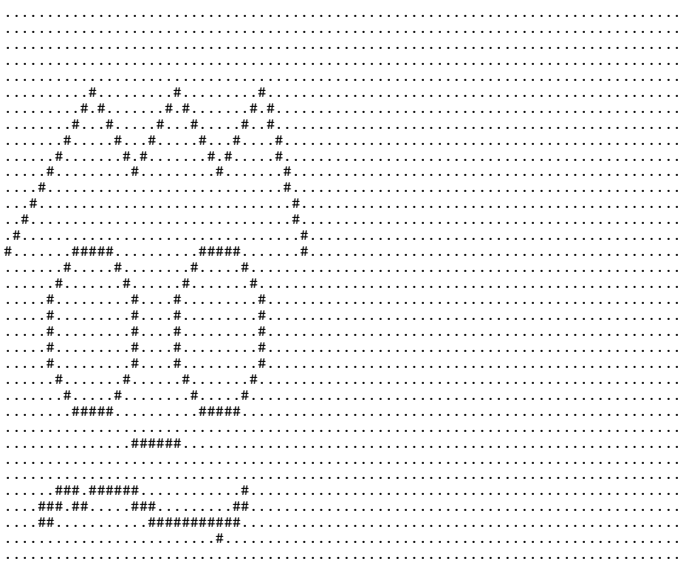
To draw the different geometric shapes on the screen, there are drawing algorithms for each basic shape. These algorithms generate an approximation of the desired shape based on its parameters.

The details of each of these algorithms are described in part 2.

Example of a generated drawing:

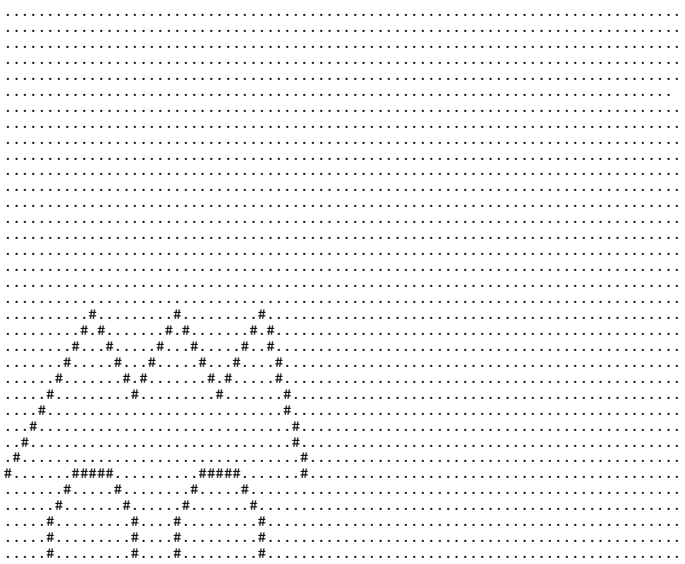
```
Please select one action:
    A- Add a shape
    B- Display the list of shapes
    C- Delete a shape
    D- Drawing the shapes
    E- Help
    [Other actions]
>> Your choice: B
    List of shapes :
        1 : CIRCLE 20 10 5
        2 : CIRCLE 20 25 5
        3 : LINE 5 20 10 25
        4 : POLYGON 15 0 5 10 10 15 5 20 10 25 5 30 15 35
        5 : CURVE 35 5 25 5 40 30 30 30
```

```
>> Your choice: D
```



Note that geometric shapes created from coordinates that are partially or totally outside the drawing area are accepted. These will be stored in memory but will not appear on the screen. In case only a part of the shape is outside the drawing area, it is necessary to display on the screen the portion of the drawing that is inside the drawing area.

Example : Same figure drawn with different coordinates



Part I.
Shapes data structures

To begin this work, we will first develop the part of the application code that allows you to create and store specific shapes: point, segment, circle, square, rectangle and polygon. We will therefore not be interested in the notion of layers, windows or drawing shapes on the screen for the moment. As we will see in more detail in section 4, we will also have to create an abstract (generic) type for managing these shapes. This generic type will be independent of a particular shape in order to facilitate the developments that will follow in the second part.

In this part we are going to create a module composed of *shapes.c* and *shapes.h* files. This module will group the structured types of each shape as well as the associated functions: creation, display and destruction. We refer in this part to the display functions allowing to display the information related to each of the shapes and not to the drawing of these shapes on the screen.

3. The specific structures of the available shapes

3.1. `Point` structure

A point is a graphic shape that represents a position in space. Our vector drawing application being textual we will use a fill character to represent it in our 2 dimensional space. In the rest of the document, we will use the '#' as a placeholder.

The representation in memory of a point will be done by a variable of type structured `Point` composed of two integer fields `pos_x` and `pos_y` representing its coordinates on the x and y axes.

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

To manipulate a point, the following functions must be defined:

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

Where:

- The function `Point *create_point(int px, int py)` allows to dynamically allocate a structured Point whose coordinates are given in parameter.
- The `void delete_point(Point * point)` function allows to free the memory allocated to the point given in parameter.
- The `void print_point(Point * p)` function allows to display on the screen the information of a Point in the following form: `POINT [p(pos_x, pos_y)]`

Example of use

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

Result on screen

```
POINT 10 15
```

3.2. `Line` structure

A line is a geometric shape corresponding to a segment delimited by two points. It is represented in memory as a structure composed of two fields of type *Point* representing the ends of the segment. Since *Point* variables are created dynamically, the fields of the structure must be of type *Point**.

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

The functions associated with this structured type are the following:

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

Where :

- The function `Line *create_line(Point * p1, Point * p2)` allows t o dynamically allocate a segment of structured type *Line* from two points given in parameters.
- The `void delete_line(Line * line)` function allows to free the memory allocated to the segment given in parameter.
- `void print_line(Line * line)` allows to display the information of a segment according to the format described in the following example: `LINE [x1, y1, x2, y2]`

Example of use

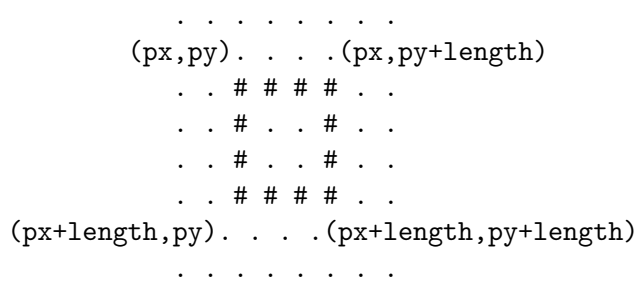
```
Point * p1 = create_point (10, 15);
Point * p2 = create_point (21, 25);
Line * l = create_line (p1 ,p2);
print_line (l);
delete_line(l);
delete_point(p1);
delete_point(p2);
```

Result on screen

```
LINE 10 15 21 25
```

3.3. `Square` structure

The square shape is represented by a point in space (upper left corner) and a length. From this information you can define the *Square* structure yourself. The positions of the other 3 corners can be calculated as follows:

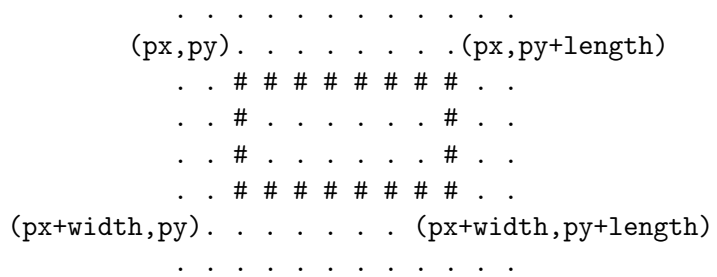


A square is thus represented in memory as a structure with two fields: a point and a length. In the same way as for the previous structures, it is necessary to implement three functions associated with the manipulation of this structured type:

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

3.4. `Rectangle` structure

The rectangle shape is represented by a point (upper left corner), a length and a width. From this information you will define the *Rectangle* structure yourself. The positions of the other 3 corners can be calculated as follows:



As with the previous structures, it is necessary to implement three functions for handling this structured type:

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

3.5. `Circle` structure

The Circle shape is represented by a point and a radius. From this information you will define the *Circle* structure yourself. In the same way as for the previous structures, it is necessary to implement three functions associated with the manipulation of this structured type:

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

3.6. `Polygon` structure

The polygon is represented as a set of points to be connected. The structured type of this shape consists of two fields: a dynamic array of points and its size n. The array of points must be dynamic (logical size equal to the physical size) because the size n is not known at compile time, it is the user who will specify it with the command to create a polygon.

Note: It is important to remember that in this project variables of type *Point* are created dynamically by the *create_point* function which returns a variable of type *Point**. The second field of the *Polygon* structure must be declared as a dynamic 1D array where each cell is of type *Point**. This field is therefore of type *Point***.

NB: To have a closed polygon, it is necessary that the coordinates of its first point are the same as those of the last point.

The structure of a polygon is as follows:

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

The functions associated with it are again similar to what was done for the previous structures:

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

4. Generic structure : `Shape`

We now have a set of specific functions to manage each of our shapes. However, the layer system presented in part 1.1 on page 5 must be able to group shapes of different types. Moreover, each shape must be uniquely identified in the application. This number must be independent of the shape but also of the layer to which the shape belongs because the user must be able to switch a shape from one layer to another.

The application must therefore be able to manipulate an abstract type that can represent any type of shape: *Point*, *Line*, *Square*, etc. This is called **genericity** in programming. In C language this genericity is obtained with a particular type of pointer : *void **.

Additional information on the `void*` type

As a reminder, a pointer is a variable used to store the address of another variable in the program. The pointer points or references another variable. The dereference operator "*" applied on a pointer allows to recover the value of the referenced variable. This dereferencing operator only works if the type of the referenced variable is known: this is why it is mandatory to specify the type of the referenced variable when declaring a pointer.

The *void** type is a so-called **generic** pointer type: it allows to store the address of a variable whose type is not known. **The dereference operator on this type of pointer is illegal** because the machine does not know how many bytes to read at this address to get the value, nor the binary encoding convention used elsewhere (cf. the numerical system module!).

You have already used generic functions using the *void** type: *printf* and *malloc*.

The function `malloc` allows to dynamically allocate any type of variable or array. There is only one version of the function `malloc` whatever the type of what is allocated (variable of type `int`, 1D array of `char`, 2D array of `float`, etc). The genericity of this function is obtained by the type of the return value which is `void*`. The function thus returns the address of a memory zone of any type. As you know, this return value must be **cast** (implicitly or explicitly by the programmer) to a new pointer of the expected type to be used in the rest of the code.

The function `printf` allows with the format `%p` to display any type of address. In this case, the genericity of the function is obtained by using a parameter of type `void*`. When the function is called, the pointer or the address specified as an argument is implicitly cast to a `void*` in order to be displayed.

What you should know about *void** pointers:

- this type of pointer allows to point to any type of variable (basic types, struc- turated, etc);
- this type is used to make functions generic in terms of their inputs and/or outputs, i.e. independent of a particular type.
- Pointer arithmetic does not exist on this type of pointer. It is mandatory to cast the pointer (implicitly or explicitly) to the desired type so that a dereferencing operation is possible for example.
- Beware of the ambiguity of the keyword `void` the expression `void*`. A pointer of this type does not point to a variable of type `void` and this expression should not be interpreted as a pointer to "nothing" either.

4.1. Structured type `Shape`

To handle this genericity for shapes we will create a structured type `Shape` using the particular type `void*` for the field to point to any type of structure that corresponds to the shapes we defined in the previous section. An additional field will represent the identification number (named `ID` in the following) to uniquely identify the shape in the application.

Note : This generic type gives the possibility to add a new form to our application while limiting the impact on the code already written. This type of benefit corresponds to the indicator called maintainability for the software quality.

Thus, our new structured type Shape will be composed of an identifier (id), a generic pointer to any shape as well as the type of the pointed shape (`SHAPE_TYPE shape_type`).

Since there is a limited list of geometric shapes, we will use an enumeration for their types.

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

typedef struct shape {
    int id;                  //  unique identifier of the shape
    SHAPE_TYPE shape_type;   // type of the shape pointed
    void *ptrShape;         // pointer to any shape
}Shape;
```

In order to manipulate these shapes in a generic way, we need to implement the following functions:

```
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);
```

The idea is that the function `Shape *create_empty_shape(SHAPE_TYPE shape_type)` allocates the memory area that will contain a shape type given in parameter. To do this, we propose to write it as follows:

```
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;
}
```

Then, create a generic function for each shape type. We propose the code of the `Shape *create_empty_shape(SHAPE_TYPE shape_type)` function below:

```
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;
}
```

It is thus asked to understand and copy the two functions provided and then to deduce the code of the following functions:

- `Shape *create_line_shape(int px1, int py1, int px2, int py2)` allows you to create an empty Shape, and then associate it with a line shape;
- The function `Shape *create_square_shape(int px, int py, int length)` allows to create an empty Shape formr, then associates it a shape of type square ;

- The function `Shape *create_rectangle_shape(int px, int py, int width, int height)` allows you to create an empty Shape, then associate a rectangle shape to it;
- The function `Shape *create_cercle_shape(int px, int py, int radus)` allows to create an empty Shape, then associate it a circle shape
- The function `Shape *create_polygon_shape(int lst[], int n)` allows to create an empty Shape, then associates it a shape of type polygon. This function must also check that the number of points n is a multiple of two, because the array must contain an even number of elements;
- The `void delete_shape(Shape * shape)` function allows to delete a shape;

- The function `void print_shape(Shape * shape)` allows to display on the screen the id of the shape and its type, then call the function print...*oftheencapsulatedshape*(*seetheexamplebelow*).

Example of use

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

Result on screen

4.2. Management of unique identifiers

Shapes are identified by a unique number. To guarantee this uniqueness, the method to be used is the following:

1. Create a global variable **global_id** initialized to 0
2. Increment the value of this variable at each creation of a geometrical shape. It is thus modified in all the functions of creation of the shapes whatever their types.

We suggest you do it as follows in `id.h` and `id.c` files.

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

- The `unsigned int get_next_id()` function increments the counter and returns its last value. Note that the first form created will have the *id* value 1.

- On **04/23/2023 at 11:59pm**, all teams must be able to upload a **.zip** archive to Moodle containing features A and B from the menu shown in section:
  1. . The addition of geometric shapes to the memory storage structure based on the proposed structures and function prototypes;
  2. Textual display (without drawing) of the geometric shapes and the list of stored shapes.
- The archive to be submitted must contain:
  - **.c** and **.h** files
  - A small report explaining the global skeleton of the first part of the project: Diagram of the structure used for the storage in memory as well as the order of the functions (menu structure in text or flowchart format)

If you see this message, it means that you have read everything, Bravo ! Keep going on this part to get the next one ☺