Introduction aux algorigrammes

par Véronique Bondaz Pascal Bouron Troumad alias Bernard SIAUD (Page personnelle)

Date de publication : 25/06/07

Dernière mise à jour : 04/07/07

Les premiers algorigrammes, les premières boucles

Avant-Propos

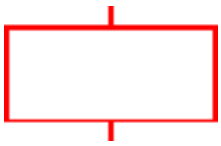
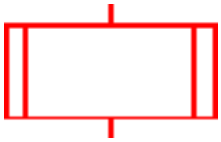
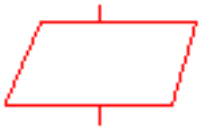
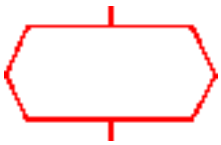
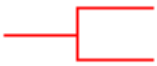
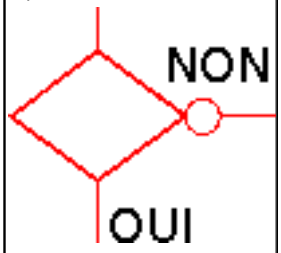
Avant toute programmation, il est recommandé d'avoir une visualisation du programme qu'on va faire. Pour cela, il faut faire un algorithme ou un organigramme. Le premier a une structure linéaire comme un programme alors que le second permet de bien mieux visualiser les différents blocs du programme, les boucles, les tests. C'est ce dernier point que je vais présenter ici : les algorigrammes.

Faire un organigramme est important car la programmation est un processus itératif. Le programme est parfois modifié par d'autres développeurs que ceux qui l'ont conçu. Ce schéma pourra expliquer la conception du programme aux nouveaux développeurs. Il pourra même éclairer le concepteur lui-même sur des idées qu'il avait eu. La réalisation d'un organigramme est tout aussi important que de mettre des commentaires dans le programme.

Les modes de programmation visuelle, qui se développent de plus en plus ressemblent plus à des algorigrammes qu'à un programme. Il est donc important de prendre connaissance dès que possible avec cette schématique.

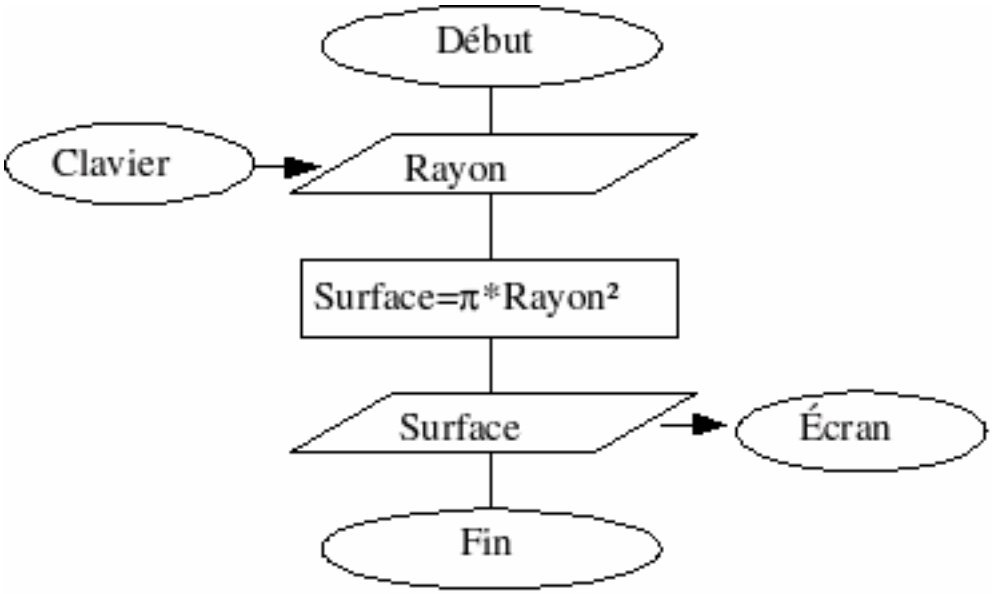
Cet article est un résumé du cours donné en II1 au département **GEii** de l'**IUT B** de l'**université Lyon 1**.

1 - La norme ISO 5807

Symbole	Désignation	Symbole	Désignation
1) 	**SYMBOLES DE TRAITEMENT** **Symbole général "traitement"** Opération ou groupe d'opérations sur des données, instructions, etc.., ou opération pour laquelle il n'existe aucun symbole normalisé.	6) 	**Mode synchrone ; mode parallèle** Ce symbole est utilisé lorsque plusieurs instructions doivent être exécutées simultanément.
2) 	**Fonction ou sous-programme** Portion de programme considérée comme une simple opération.	7) 	**SYMBOLES AUXILIAIRES** **Renvoi** Symbole utilisé deux fois pour assurer la continuité lorsqu'une partie de ligne de liaison n'est pas représentée.
3) 	**Entrée - Sortie :** Mise à disposition d'une information à traiter ou enregistrement d'une information traitée.	8) 	**Début, fin, interruption** Début, fin ou interruption d'un organigramme, point de contrôle, etc..
4) 	**Préparation** Opération qui détermine partiellement ou complètement la voie à suivre dans un embranchement ou un sous-programme. Symbole également utilisé pour préparer une décision ou mettre un aiguillage en position.	9) 	**Commentaire** Symbole utilisé pour donner des indications marginales.
3) 	**SYMBOLES LOGIQUES** **Embranchement** Exploitation de conditions variables impliquant le choix d'une voie parmi plusieurs. Symbole couramment utilisé pour représenter une décision ou un aiguillage.	**Sens conventionnel des liaisons :** Le sens général des lignes doit être : - de haut en bas - de gauche à droite. Lorsque le sens ainsi défini n'est pas respecté, des pointes de flèches, à cheval sur la ligne, indiquent le sens utilisé.	

Exemple : Calcul de la surface d?un disque à partir du rayon

Traduction du cahier des charges :
Entrée : Saisie du rayon
Sortie : Affichage de la surface du disque
Traitement à réaliser : Surface=Pi* Rayon²



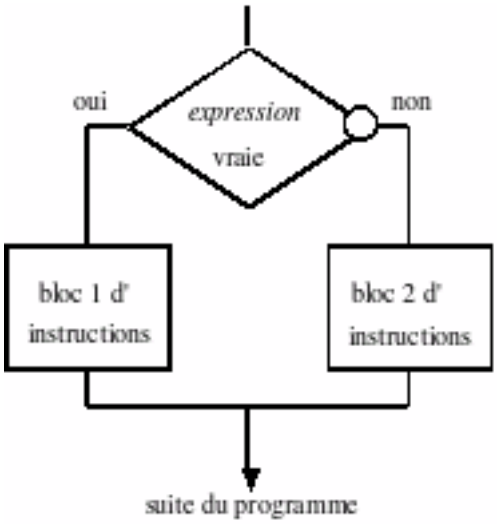
2 - Les entrées-sorties

Les entrées sorties sont schématisées par des flèches. Elles sont représentées, par une flèche qui va vers une bulle ou un texte pour les entrées et, par une flèche qui sort d'une bulle ou d'un texte pour les sorties. Pour plus de lisibilité, on choisit de mettre à gauche les entrées et à droite les sorties. La nature des entrées-sorties est aussi indiquée car elles peuvent être complètement différentes surtout en automatique.
Voir sur le schéma ci-dessus l'éntrée **Clavier** et la sortie **Écran**.

3 - Structure d?aiguillage : Si (...) {...} Alors {...}

Il s'agit de l'instruction :

```
SI ( expression vraie ) ALORS
    BLOC 1 D'INSTRUCTIONS
SINON
    BLOC 2 D'INSTRUCTIONS
FINSI
```



remarque : la sortie avec le rond est la sortie « non ». Ceci permet de la mettre n'importe où : en bas, à droite ou à gauche.

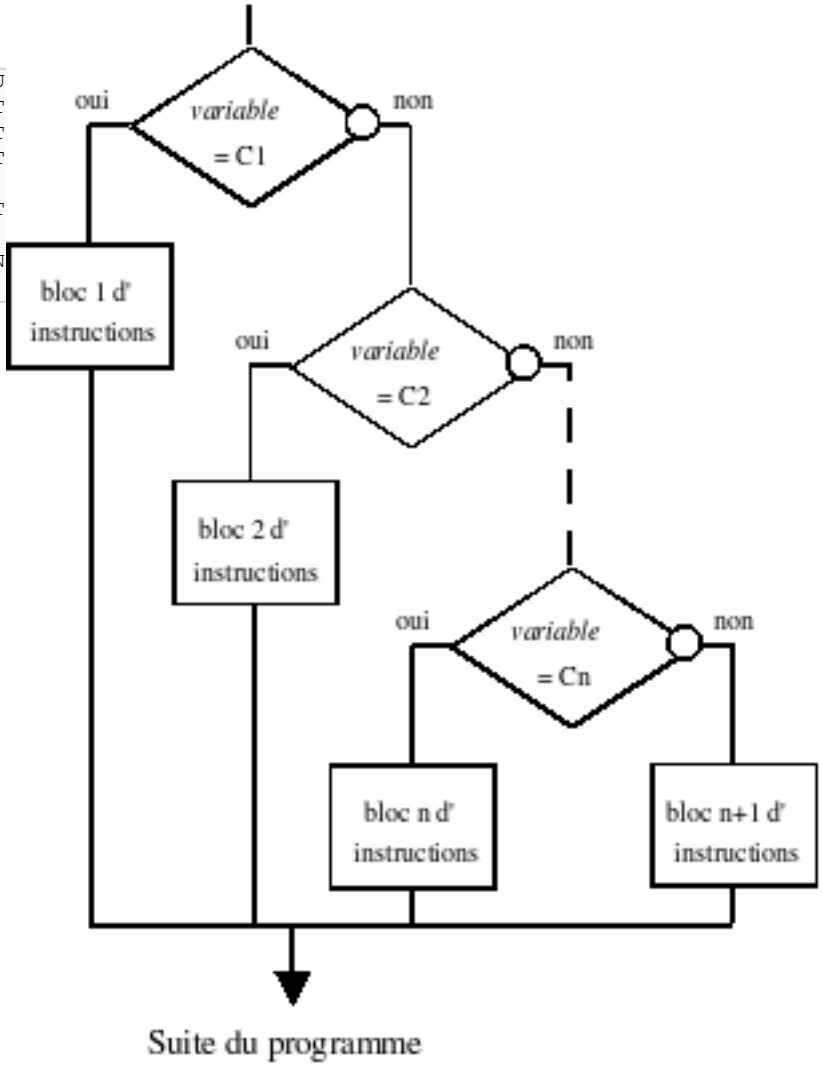
4 - Structure de choix multiples : Au cas ou...

La structure Si (...) Alors {...} Sinon {...} permet de réaliser un choix parmi deux possibilités. Il est possible d?imbriquer les Si (...) Alors {...} Sinon {...} les uns dans les autres pour tenir compte de choix plus nombreux. Mais la structure devient très lourde à écrire et à lire quand le nombre de cas augmente. On lui préférera alors la structure de tests multiples, qui permet de comparer une variable (de type entier en C) à toute une série de valeurs et d?exécuter, en fonction de la valeur effective de cette variable, différents blocs d?instructions.

Le code compilé par une suite de **si** ou un **Au cas ou** n'est pas le même. Celui du **Au cas ou** est plus optimisé en général.

Il s'agit de l'instruction :

```
AU CAS OU ( la variable ) VAU
       C1 : BLOC 1 D'INSTRUCT
       C2 : BLOC 2 D'INSTRUCT
       C3 : BLOC 3 D'INSTRUCT
       ?.
       Cn : BLOC n D'INSTRUCT
SINON
       BLOC n+1 D'INSTRUCTION
FIN DE CAS
```

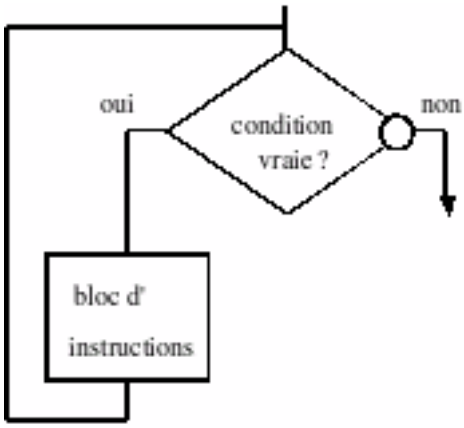


5 - Les boucles tant que et faire...tant que

5.1 - La boucle " tant que "

Il s'agit de l'instruction :

```
TANT QUE ( condition est vraie ) FAIRE
BLOC D'INSTRUCTIONS
```



Remarque : comme le test se fait avant le bloc d'instructions, **celui-ci n'est pas forcément exécuté.**

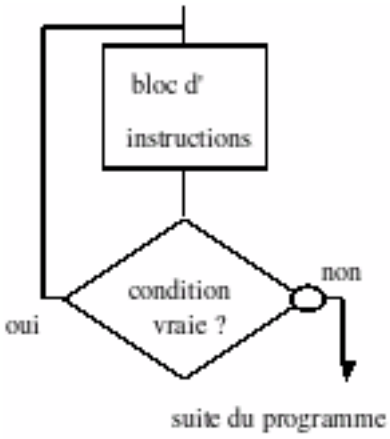
Attention : il faut que le résultat du test puisse être modifié d?une manière ou d?une autre si on ne veut pas faire une boucle sans fin.

On peut rencontrer la construction **tant que (expression);** sans la présence du bloc d'instructions. Cette construction signifie: **"tant que l'expression est vraie attendre".**

5.2 - La boucle " repeter tant que "

Il s'agit de l'instruction :

```
REPETER
        BLOC D'INSTRUCTIONS
TANT QUE (condition est vraie)
```



Remarque : Le test se fait après le bloc d'instructions, **celui-ci est exécuté au moins une fois**.

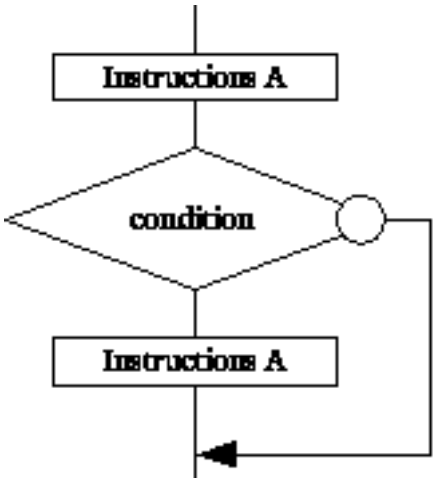
5.3 - Remplacer la boucle " repeter tant que " par "tant que"

```
REPETER
 BLOC D'INSTRUCTIONS A
TANT QUE (condition est vraie)
```

est équivalent à

```
BLOC D'INSTRUCTIONS A
TANT QUE (condition est vraie) FAIRE
 BLOC D'INSTRUCTIONS A
```

L'inconvénient de ce remplacement est la répétition de **BLOC D'INSTRUCTIONS A**. Cette remarque est juste là pour vous aider à faire la différence entre un " repeter tant que " et un "tant que", un "do while" et un "while".

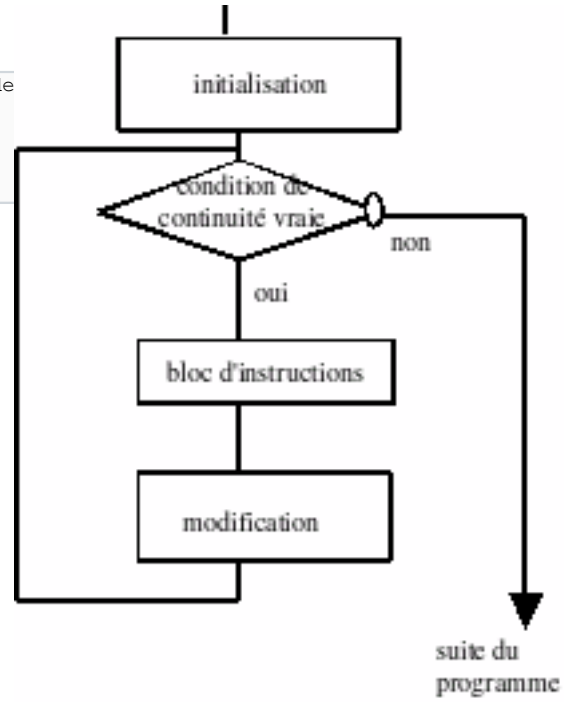


Remplacement du "faire tant que" par un "repeter tant que"

6 - L'instruction Pour

Il s'agit de l'instruction :

```
POUR ( initialisation ; condition de
 continuité vraie ; modification )
{
       BLOC D'INSTRUCTIONS
}
```



Remarques :

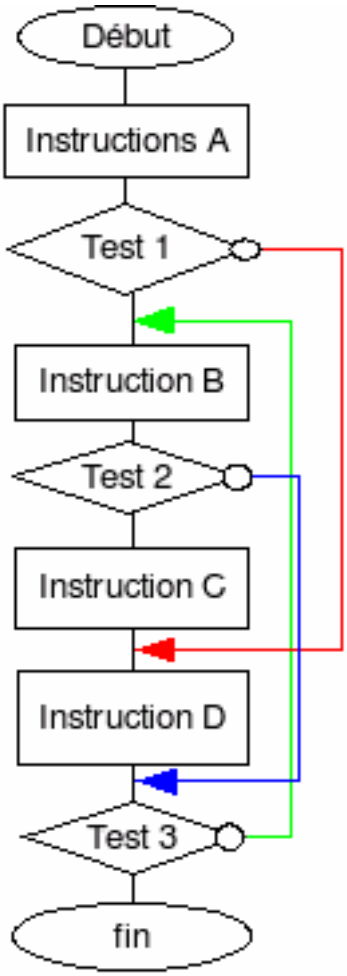
- Les 3 instructions du Pour ne portent pas forcément sur la même variable. Une instruction peut contenir l'opérateur séquentiel afin de pouvoir mettre deux instructions à la place de l'initialisation, la condition de continuité ou la modification. Attention, ceci affecte gravement la lisibilité du code.
- Une ou plusieurs des 3 instructions peuvent être omises, mais pas les ;

  Pour (;;) est une boucle infinie (répétition infinie du bloc d'instructions).

7 - Les structures à éviter - Exemple et contre exemple

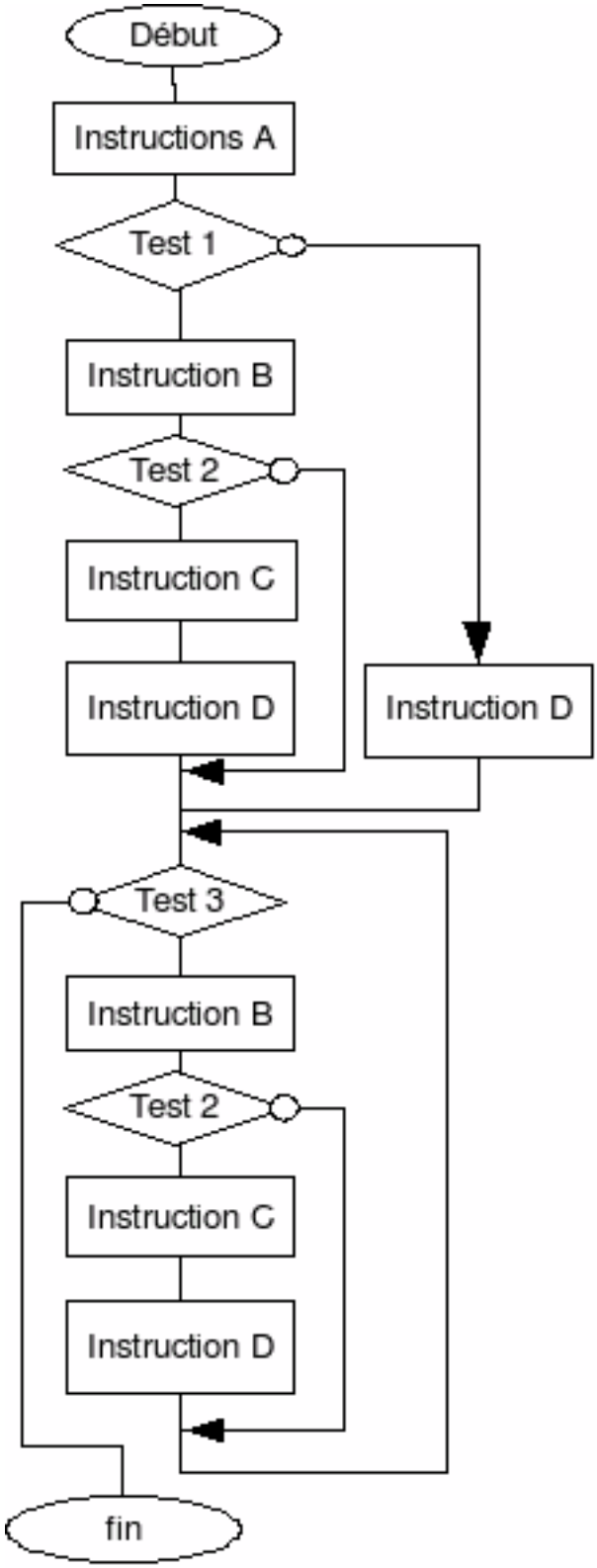
Bien que les algorigrammes soient très flexibles, leur lisibilité impose d'éviter d'utiliser des structures comme des renvois croisés (boucles ou sauts). Ces renvois doivent impérativement être contenus les uns dans les autres sans intersection.

7.1 - algorigramme incorrect



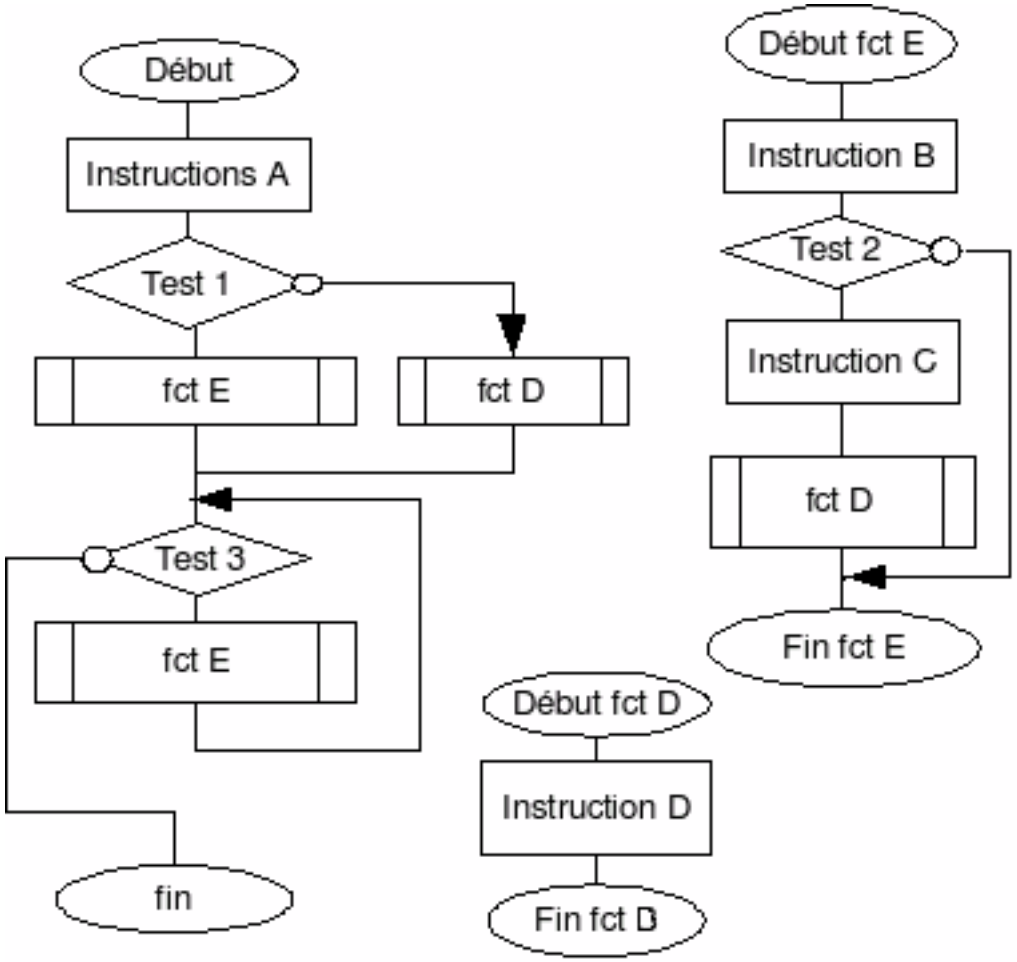
Ceci est un mauvais exemple car les trois renvois se mélangent.

7.2 - algorigramme corrigé



Cet algorigramme est correct mais, comme certaines parties sont écrites deux fois, il peut être simplifié.

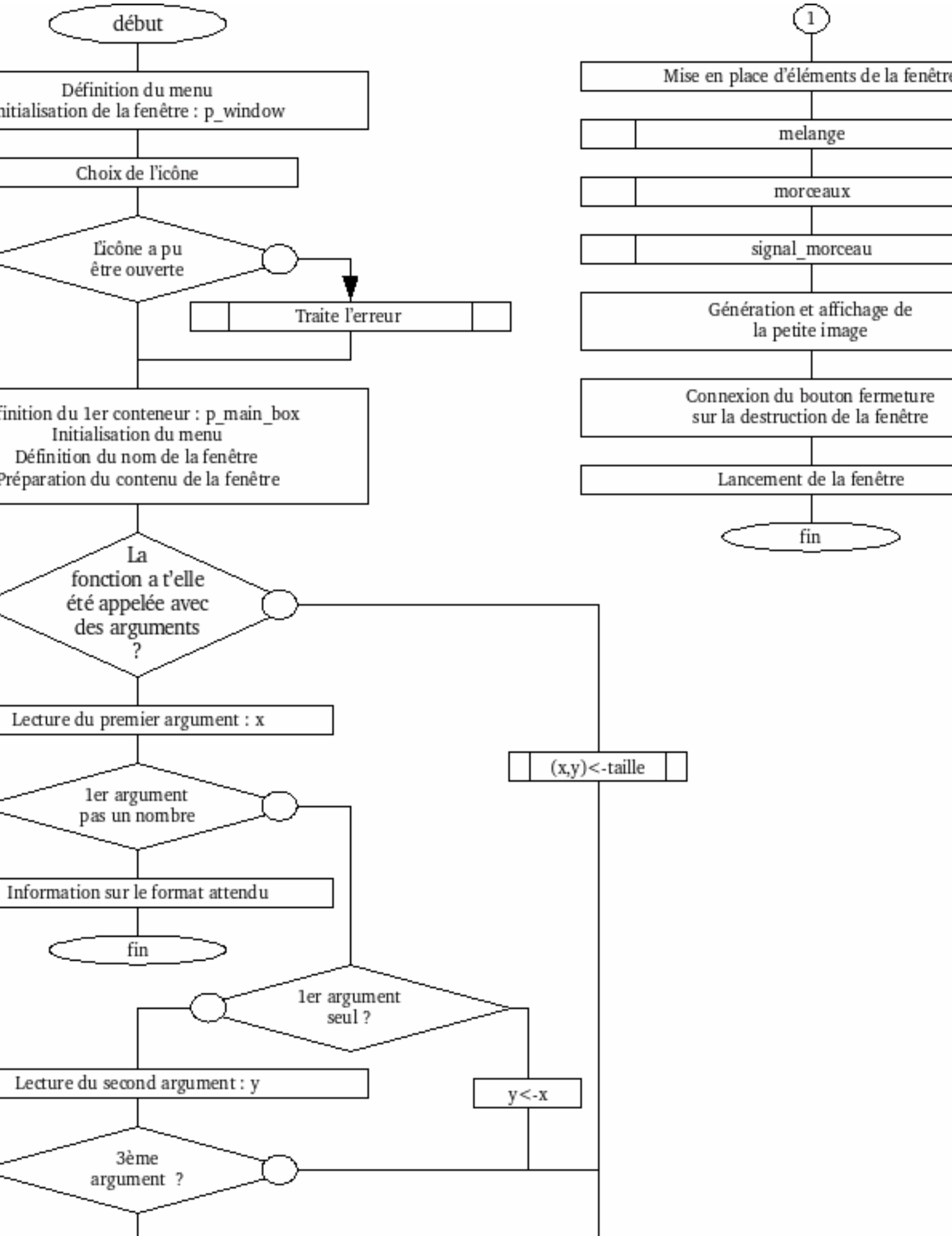
7.3 - algorigramme correct simplifié



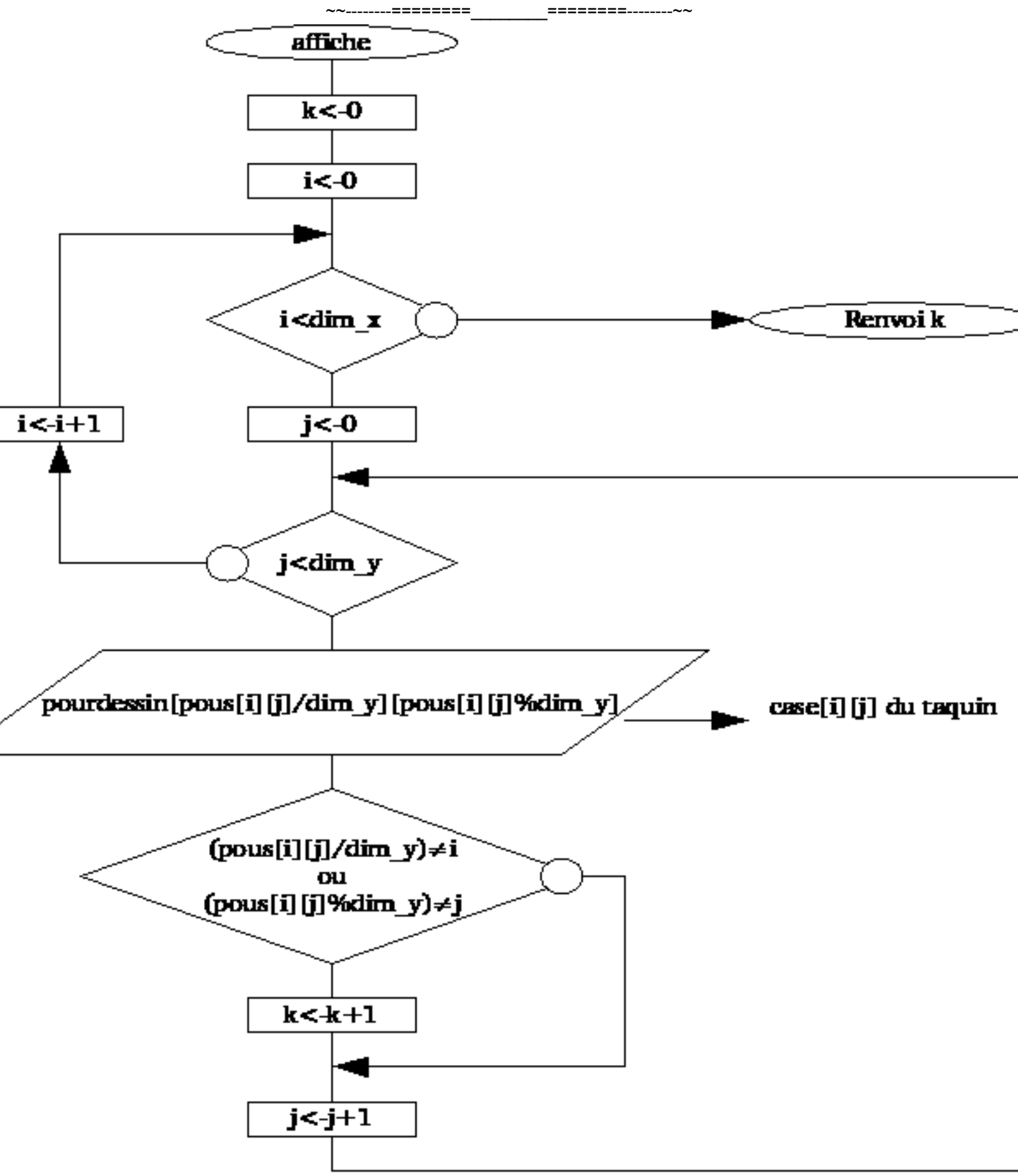
Grâce aux fonctions, la structure du programme principal a été simplifiée en évitant de faire plusieurs fois les mêmes choses.

8 - Un exemple complet

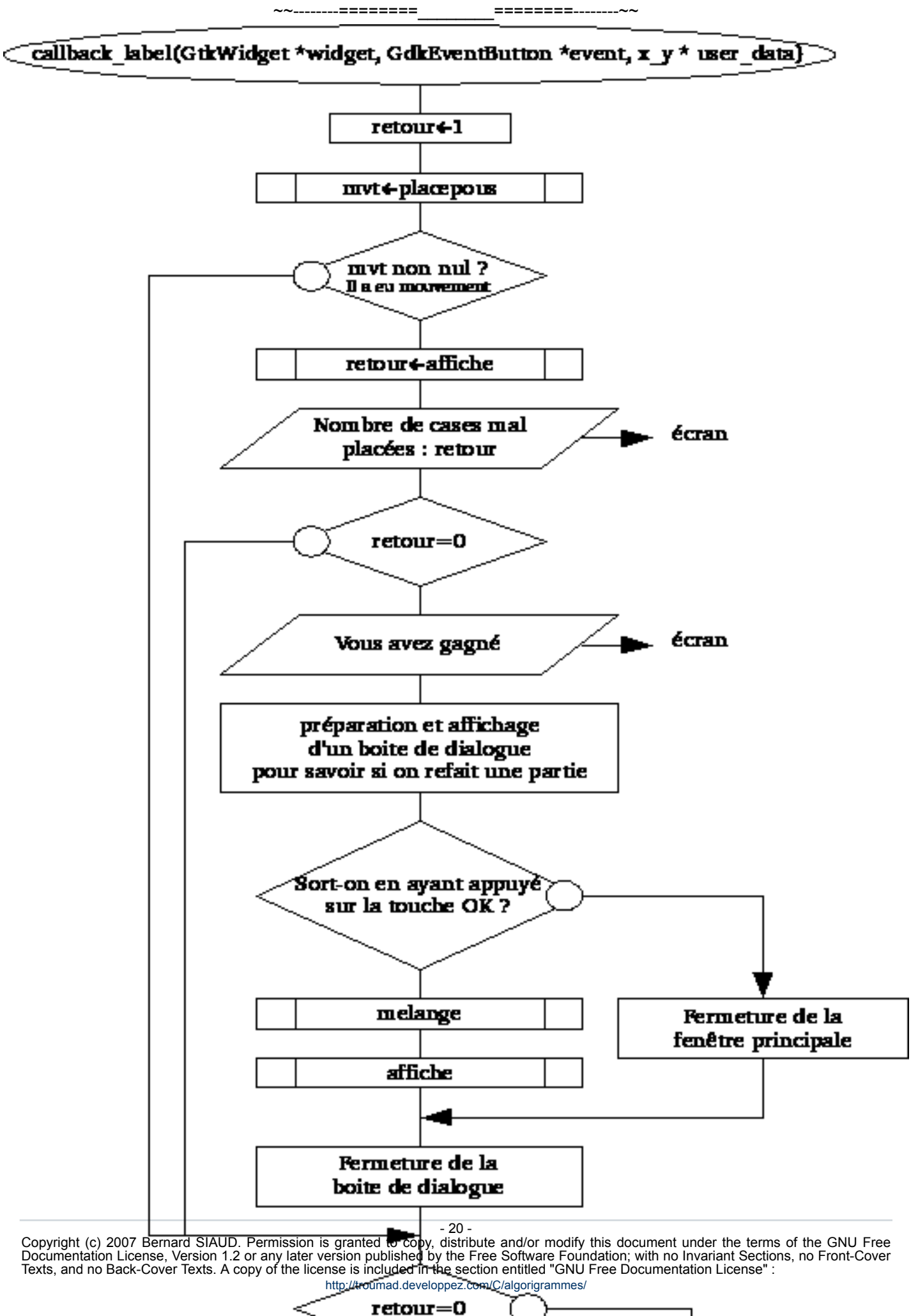
Voici, sous forme d'organigramme mon programme de taquin disponible à **http://c.developpez.com/sources/c/?page=IX**. L'original des algorigrammes est un fichier draw de **OpenOffice.org** disponible **ici**.


début
Définition du menu
itialisation de la fenêtre : p_window
Choix de l'icône
l'icône a pu
être ouverte
Traite l'erreur
finition du 1er conteneur : p_main_box
Initialisation du menu
Définition du nom de la fenêtre
Préparation du contenu de la fenêtre
La
fonction a t'elle
été appelée avec
des arguments
?
Lecture du premier argument : x
1er argument
pas un nombre
Information sur le format attendu
fin
1er argument
seul ?
Lecture du second argument : y
3ème
argument ?
(x,y)<-taille
y<-x
1
Mise en place d'éléments de la fenêtre
melange
morceaux
signal_morceau
Génération et affichage de
la petite image
Connexion du bouton fermeture
sur la destruction de la fenêtre
Lancement de la fenêtre
fin

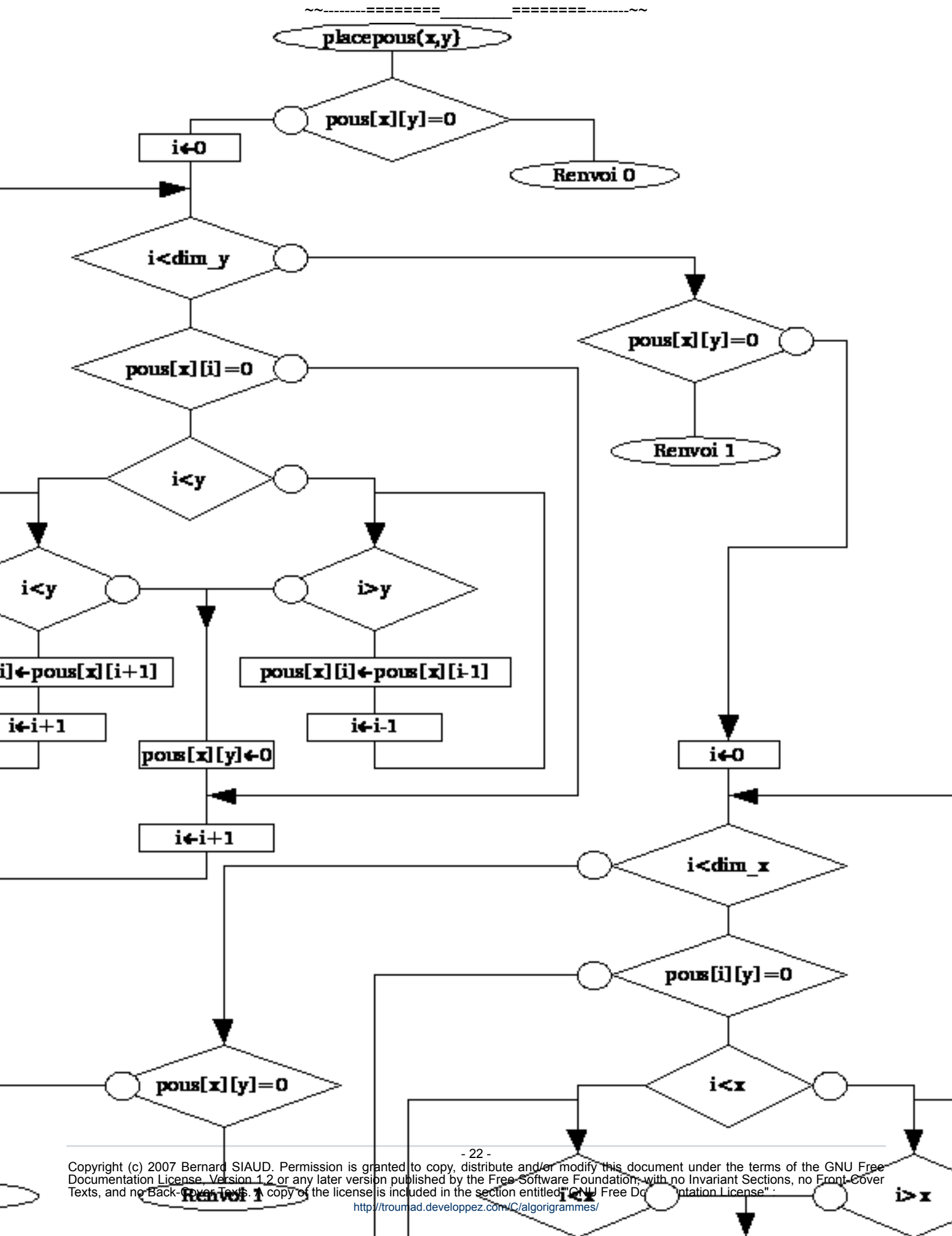
Le corps du programme



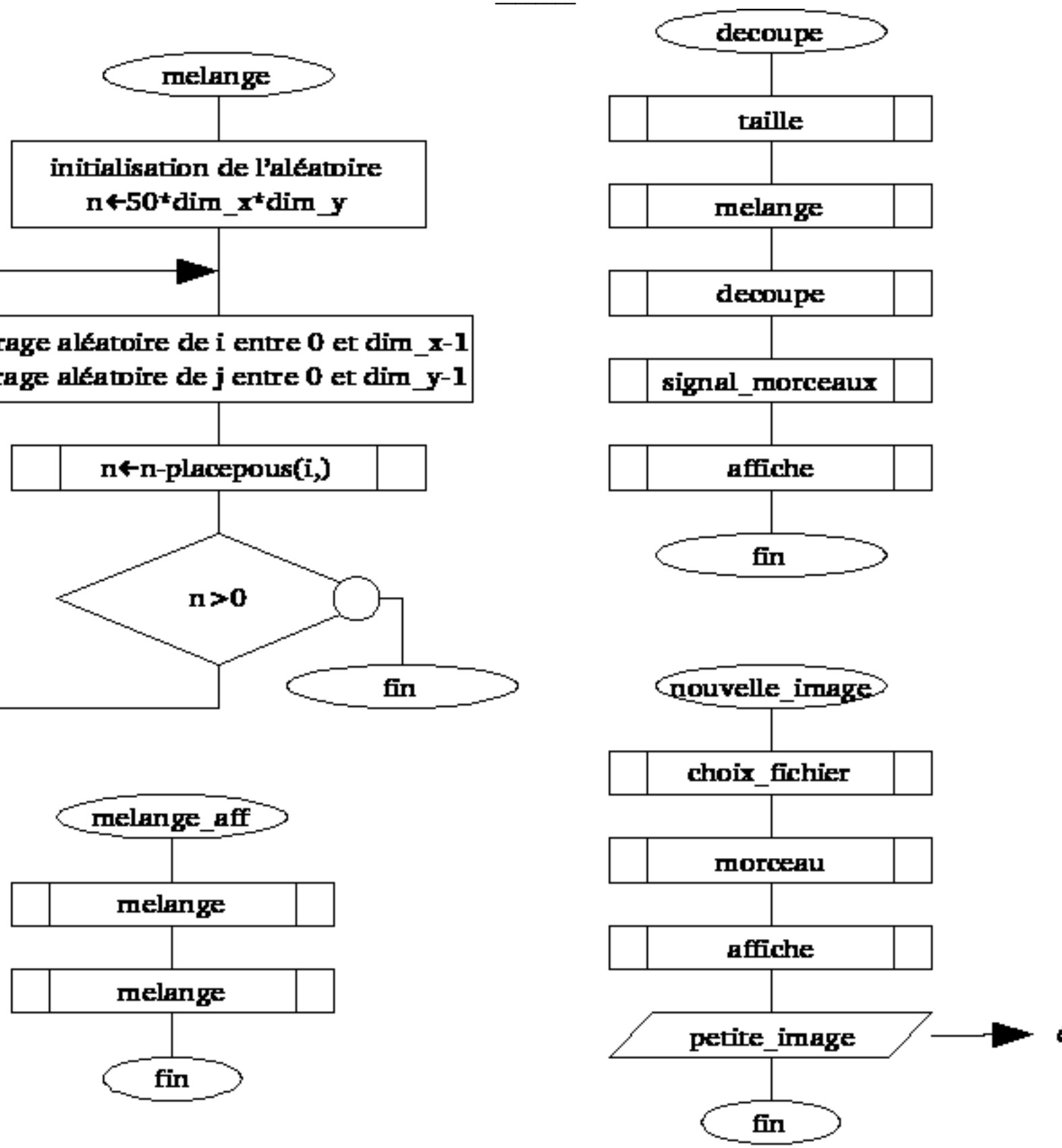
La fonction affiche qui renvoie le nombre k de cases mal placées.



La fonction callback_label qui renvoie 1 si le taquin est fini, 0 sinon.

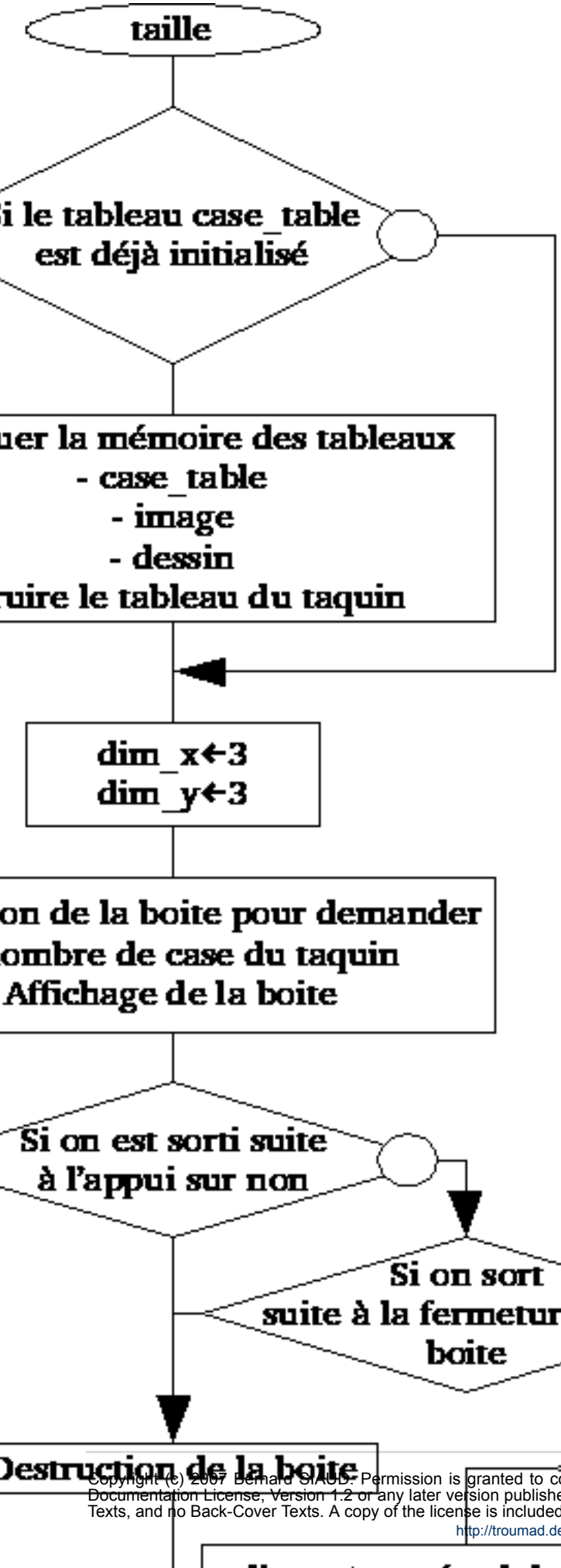


La fonction placepous qui renvoie 1 s'il y a eu un mouvement.

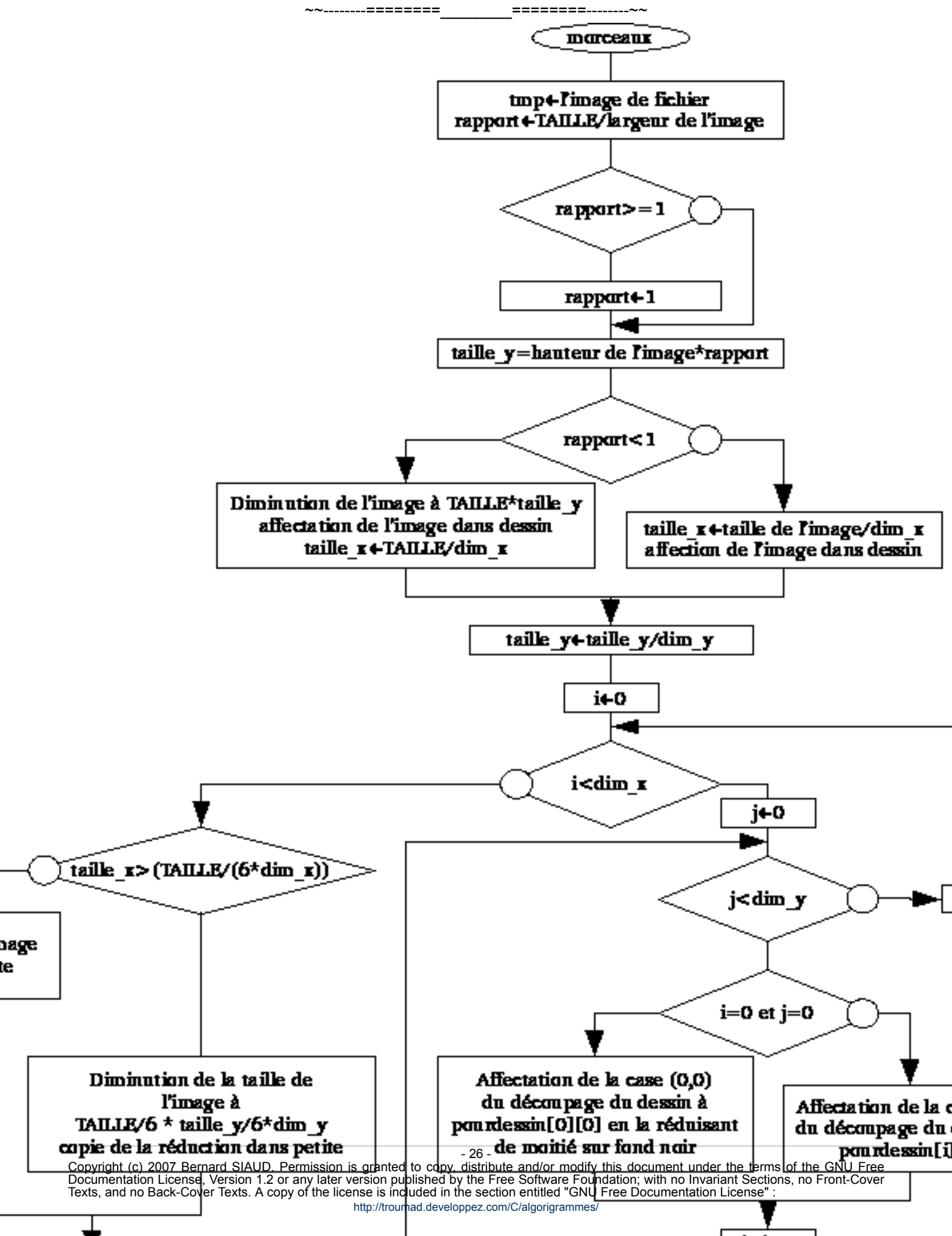


4 petites fonctions.

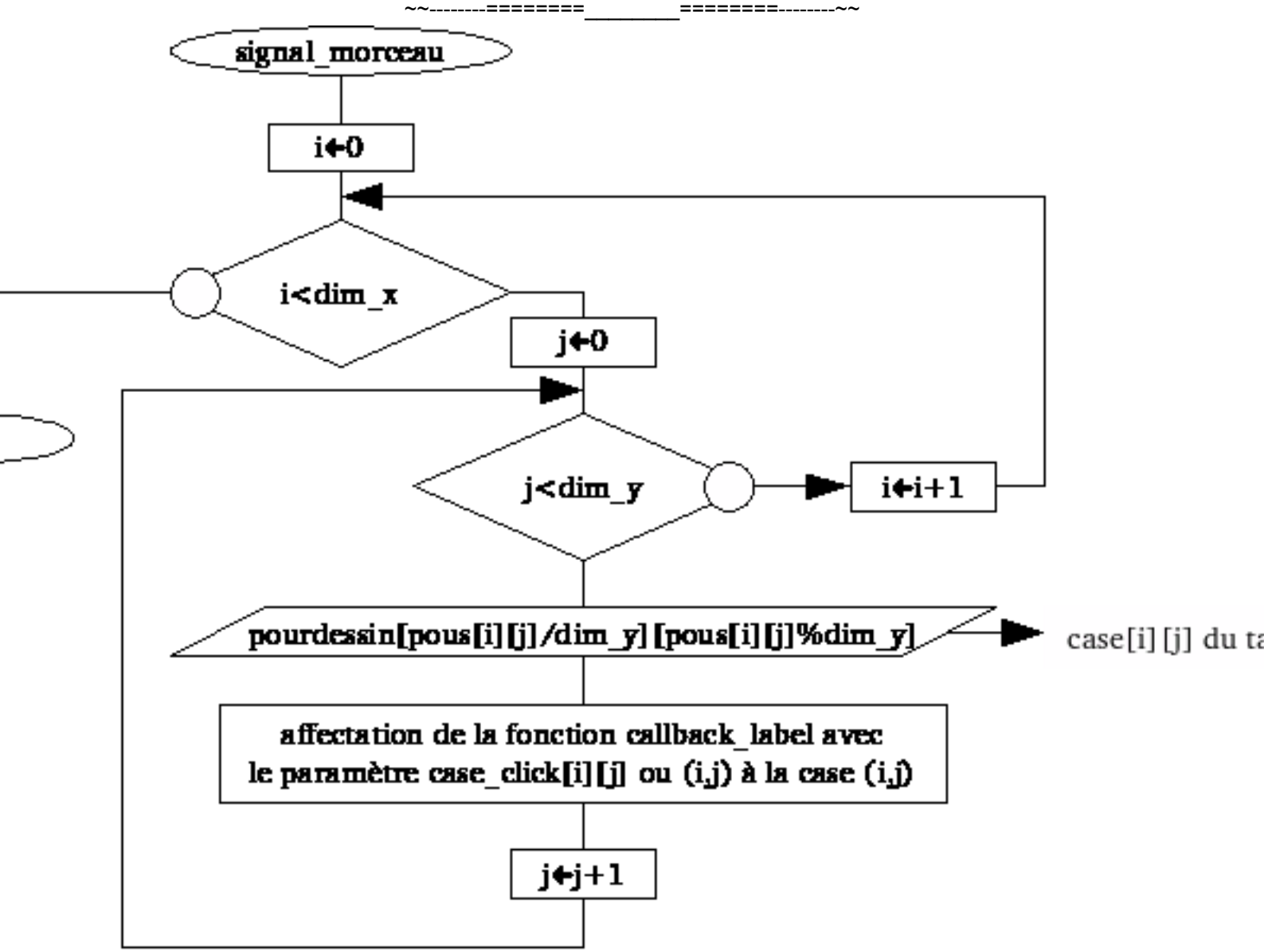
~~--------========_____========--------~~



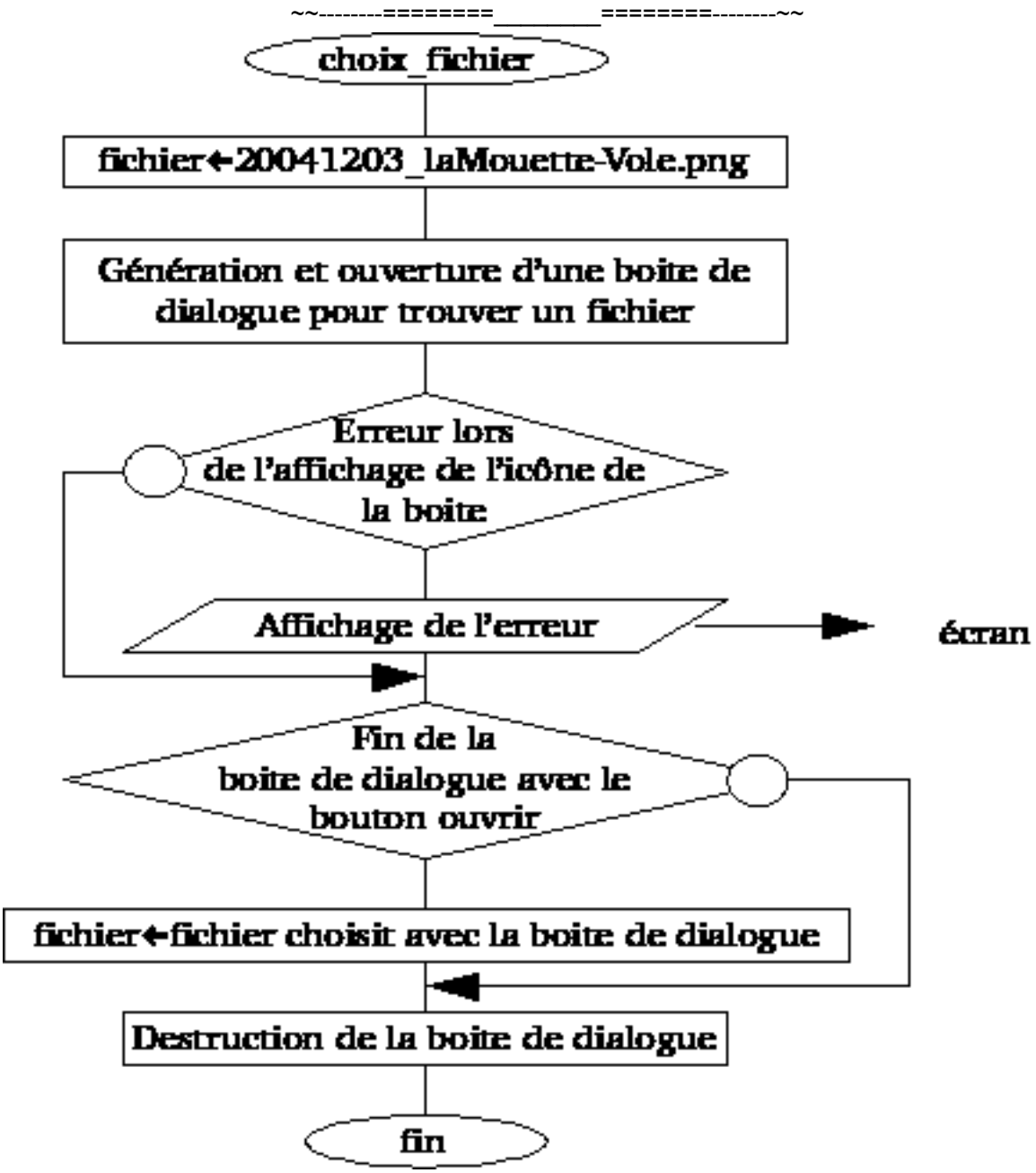
La fonction taille



La fonction morceaux



La fonction signal_morceaux



La fonction choix_fichier

9 - Remerciements

PRomu@ld pour ses conseils et **Dut** pour sa correction.