Marie Paule Bwelle
17802001

Amina Boughatane
17808147

PROJET N°30 GRILLES DE NOMBRES

2019/2020

Table des matières

I. Introduction

Dans le cadre de notre projet, nous avons choisi de réaliser le projet numéro 30, grilles de nombres croisés. Il consiste à programmer en C une grille de nombres croisées à tailles variées choisis par un utilisateur. Le rôle de l'utilisateur est de remplir la grille en choisissant une des définitions proposées (carré, nombre premier…), il peut aussi mettre en place une case noire dans une des cases de la grille, cette case sera bloquée et le programme ne pourra pas ajouter de valeur à cette case. Nous avons choisi ce projet principalement par intrigue. Il nous rappelait les énigmes mathématiques auquel nous avions droit au collège. Les débuts furent étrangement compliqués. Simple de prime abord, ce projet était difficile à comprendre. Nous avons eu énormément de mal du début à la fin, nous avons même dû « recommencer » entièrement notre projet quelques jours avant sa remise, car nous étions complètement perdus.

II. Explications sur les procédures mises en œuvre

1. Fonctions pour l'utilisation de la grille

Avant tout, nous devions mettre en place une grille pour y travailler, ce que nous avons fait avec ces fichiers :

initialisations.c (Amina et MP) : Ce fichier contient toutes les fonctions d'initialisation. Il est composé de 3 fonctions :

*void initGrid(champ_t * grid, int w, int h);*

- Cette fonction nous permet d'initialiser notre matrice à une certaine valeur. Nous avons choisi le numéro 10 car celui-ci n'est pas un chiffre et donc ne sera pas utilisé pour remplir notre grille. On parcourt alors toute notre matrice en remplissant chaque case.
- grid est la grille à initialiser.
- w la largeur choisie par l'utilisateur.
- h la hauteur choisie par l'utilisateur .

*void initBits(champ_t * grid, int w, int h) ;*

- initBits se charge de l'initialisation des bits d'un champ à zéro pour avoir la possibilité d'autoriser des chiffres si besoin.
- grid est notre grille.
- w et h sa largeur et sa hauteur.

champ_t * inputGrid(int input,champ_t * grid, int w, int h) ;

- On utilise cette fonction pour que l'utilisateur puisse entrer l'endroit où il souhaite placer ses définitions.
- input est l'entrée de l'utilisateur.
- grid la grille.
- w et h sa largeur et sa hauteur.
- Elle retourne l'adresse où l'utilisateur souhaite entrer sa définition.

affichage.c (Amina et MP): Dans ce fichier, nous trouvons trois fonctions d'affichage :

*void printGrid(champ_t * grid, int w, int h);*

- printGrid permet d'afficher notre grille. Pour toutes les cases qui valent 10, on affiche une case vide. Pour les cases qui valent 11, on affiche une case noire (*).
- grid la grille.
- w et h sa largeur et sa hauteur.

void affichamp(champ_t cell) ;

- Cette procédure nous permet d'afficher nos cases si nécessaire.
- cell représente la case à afficher

Void affTab(tab t) ;

- Permet d'afficher si besoin un vecteur.
- t est le vecteur à afficher.

chiffres_Interdits.c (Amina et MP) : Dans ce fichier, nous trouverons deux fonctions liées aux chiffres interdits dans une case :

*void digitAllowed(champ_t * t, int chffr);*

- digitAllowed définit les chiffres autorisés pour une case.
- t est la case pour laquelle on autorise le chiffre
- chiffr est le chiffre qu'on va autoriser

*int isAllowed(champ_t * t, int chffr) ;*

- Test si un chiffre est autorisé pour une case.
- t est la case que l'on test.
- Retourne 1 si le chiffre est autorisé, sinon 0.

case_Noire.c (MP) : Pour établir la case noire, une case qui change les dimensions d'une partie seulement de la matrice et qui donc va affecter qu'une des définitions, on utilise ce fichier. Il est composé de 2 fonctions :

*void initBlackCell(champ_t * here, champ_t ** ptrCaseNoire, int noireColonne, int noireLigne, champ_t * grid, int w, int h)*

- Cette procédure permet d'initialiser la case noire si l'utilisateur le souhaite.
- here pointe là où l'utilisateur ajoute la case noire.
- ptrCaseNoire est l'adresse mémoire de notre case noire qui nous permet de vérifier si elle se trouve sur une fonction lorsqu'on travaille dessus.
- noireColonne contient la valeur de l'emplacement de la case noire en horizontale.
- noireLigne contient la valeur de l'emplacement de la case noire en verticale.
- grid nous donne accès à notre grille de travail.
- w correspond à la taille de la grille en largeur.
- h correspond à la taille de la grille en hauteur.

*void toAvoidBlackCell(champ_t * here, champ_t * ptrCaseNoire, int input,champ_t ** ptrGrid, int * w, int * h)*

- Celle-ci est utilisé dans nos définitions carré, chiffres consécutifs et nombres premiers dans le cas où l'utilisateur place la case noire sur l'une d'entre elle et du coup affecte les dimensions de ses fonctions.
- here localise la fonction en cours de traitement par cette fonction.
- ptrCaseNoire nous donne accès à l'emplacement de la case noire.
- input correspond à l'entrée de l'utilisateur.
- On prend l'adresse mémoire de la grille pour la modifier si besoin.
- On prend l'adresse mémoire de w pour la modifier si besoin
- On prend l'adresse mémoire de h pour la modifier si besoin.

2. Fonctions pour les définitions

La grille a besoin de définition pour être résolu, nous avons donc décidé de commencer avec celles que vous nous avez proposé. Le but était de donner à l'utilisateur la possibilité parmi les choix qu'on lui offrait de résoudre un problème. Voici nos fichiers :

Chiffre_Consecutifs.c (Amina) **:** Dans ce fichier, nous trouverons la fonction *consecutiveDigit* dont le prototype est le suivant :

*Void consecutiveDigit(int * lastConsecutiveDigits,champ_t * here, champ_t * ptrCaseNoire,int input,champ_t *grid, int w, int h)*

consecutiveDigit permet de placer une série de chiffres consécutifs, elle prend en paramètres :

- Un pointeur (*int * lastConsecutiveDigits)* qui pointe sur le dernier chiffre tester dans la grille.
- Un pointeur (*champ_t * here)* qui pointe sur l'endroit où l'on souhaite placer notre élément dans la grille.
- Un pointeur (*champ_t * ptrCaseNoire)* qui pointe sur la case noire que l'utilisateur choisira où il voudrait la mettre.
- Une variable *(int input)* qui prend le choix de l'endroit entrée par l'utilisateur afin de mettre en place la définition choisie.
- *(champ_t *grid)* représente la grille du jeu.
- Deux variables *(int w) et (int h)* qui représentent la largeur et la hauteur de la grille

Au début de la fonction notre pointeur *lastConsecutiveDigits* ne pointe sur aucune variable il est donc NULL, on utilise notre variable locale « digit » qui prend la valeur de la case si elle est présente sinon la valeur 1, le pointeur gardera cette valeur, qui sera donc la dernière valeur tester pour le moment, digit incrémente pour tester la prochaine valeur qui sera à son tour la dernière valeur tester et donc remplacer l'ancienne dans le pointeur et ainsi de suite jusqu'où on arrive à la suite de chiffres consécutifs adaptés à notre grille.

La fonction *consecutiveDigit* s'adapte dans le cas de remplissage horizontal ainsi que vertical selon le choix de l'utilisateur entré dans la variable input en paramètre (1, 2 ou 3 pour les lignes) ou (a, b ou c pour les colonnes).

nombres_Premiers.c (MP) : Les fonctions de ce fichier servent à la définition de « nombres premiers », un nombre qui n'a que deux diviseurs, 1 et lui-même. Voici nos méthodes :

int findPrimeNumber(int nb);

- Cette fonction permet de trouver un nombre premier à partir d'un nombre entré en paramètre. On ajoute 1 au nombre pour avancer dans nos recherches puis on essaye de le diviser par tous les nombres. Si une de ses divisions donne un nombre pair, il n'est pas premier sinon il ne peut que se diviser lui-même et 1.
- Le nombre entré en paramètre +1 est le premier chiffre à partir duquel on cherche.

*void searchPrimeNumber(int * lastPrime,champ_t *here, champ_t * ptrCaseNoire, int input, champ_t * grid, int w, int h)*

- On utilise cette procédure pour chercher un nombre premier correspondant à l'emplacement de cette définition. Tant qu'il ne correspond pas aux chiffres autorisés par la fonction carré palindrome, on continue de chercher.
- lastPrime est le dernier nombre testé dans notre grille, il nous permet d'avancer dans la recherche.
- here localise l'endroit où placer la fonction
- ptrCaseNoire nous donne accès à l'emplacement de la case noire.

- input est l'entrée de l'utilisateur
- grid nous donne accès à notre grille de travail.
- w correspond à la taille de la grille en largeur.
- h correspond à la taille de la grille en hauteur.

carre.c (MP): Ce fichier nous permet d'utiliser la définition de « nombre carré », c'est-à-dire un nombre dont la racine carrée est un entier. Il est composé 2 méthodes :

int isSquare(int nb);

- On utilise cette fonction pour tester si le nombre est carré ou non.
- Elle prend un nombre en paramètre qu'elle va tester.
- Elle return 1 si le nombre entré est carré et 0 sinon.

*void searchSquare(int * lastSquare, champ_t * here, champ_t * ptrCaseNoire, int input,champ_t * grid, int w, int h);*

- Cette fonction nous permet de chercher un nombre carré pour notre grille. On rappelle la fonction tant que son chiffre ne correspond pas avec la définition du nombre palindrome et carré.
- lastSquare est le dernier nombre testé dans notre grille, il nous permet d'avancer dans la recherche.
- here pointe l'endroit où la définition est placée à la fin de la recherche.
- ptrCaseNoire est ce qui nous permet d'identifier la case noire. On l'utilise pour localiser l'endroit où se trouve la case noire pour modifier ou non la définition.
- input stocke l'entrée de l'utilisateur pour placer la fonction de nombre carré.
- grid nous donne accès à notre grille de travail.
- w correspond à la taille de la grille en largeur.
- h correspond à la taille de la grille en hauteur.

produit.c (Amina) **:** Dans ce fichier nous trouverons la fonction *produit* dont le prototype est le suivant :

*void produit(champ_t *here, int input,champ_t *grid, int w, int h);*

Produit permet de trouver une valeur à partir de deux valeurs déjà existantes dans la grille pour que le produit de ces dernières soit égal à 12, elle prend en paramètre :

- Un pointeur (*champ_t *here)* qui pointe sur l'endroit où l'on souhaite placer l'élément dans la grille.
- Une variable *(int input)* qui prend le choix de l'endroit entrée par l'utilisateur afin de mettre en place la définition choisie.
- *(champ_t *grid)* représente la grille du jeu.
- Deux variables *(int w) et (int h)* qui représentent la largeur et la hauteur de la grille.

Dans cette fonction on utilise trois variables locales ch1, ch2 et ch3, et un pointeur (*ptrGrid) qui au début pointera sur le même endroit que le pointeur (*here), ch1 prendra la valeur de la 1ere case remplie pointée par ptrGrid ensuite on avance notre pointeur à la 2éme case pour affecter celle-ci à notre variable ch2, on avance encore une fois ptrGrid pour arriver à la case vide qui représente ch3.

Pour trouver ch3 il suffit de diviser 12 par le produit de ch1 et ch2, si l'une des valeurs est égale à 0 la fonction enverra un message d'erreur qui plantera le programme.

Cette fonction s'adapte dans le cas de remplissage horizontal ainsi que vertical comme toutes les fonctions présentent dans notre programme.

carre_Palindrome.c (MP) : Dans ce fichier, nous trouvons plusieurs fonctions que l'on utilise pour un « nombre carré et palindrome ». Ces nombres se lisent dans les deux sens, comme par exemple 323, et leur racine carrée est un entier. Il est composé de :

int numberOfDigits(int nb);

- Grâce à celle-ci, nous pouvons obtenir le nombre de chiffres que compose un nombre. On l'utilise dans la fonction *doUpsideDown* pour avoir un arrêt lorsqu'on renvoie les chiffres du nombre.
- Elle prend en paramètre un nombre qui est séparé chiffre par chiffre pour compter combien de chiffre le compose.
- Elle renvoie le nombre de chiffres dont est composé le nombre entré en paramètre.

int doUpsideDown(int nb) ;

- Cette fonction nous est utile pour le test qui va suivre. Elle met à l'envers le nombre entré en paramètre.
- Elle retourne ensuite le nombre mis à l'envers.

int isPalindrome(int nb, int nbUpsideDown) ;

- Celle-là est un test qui vérifie si un nombre est oui ou non un palindrome.
- Elle prend en paramètre le nombre à l'endroit.
- Puis le nombre à l'envers pour les comparer.
- Elle retourne 1 s'ils sont pareil et 0 sinon.

int isPalindromicSquare(int nb) ;

- Après avoir vérifié que le nombre est un palindrome, on test si sa racine est un entier et donc que ce nombre est carré.
- Test si la racine du nombre entrée en paramètre est un entier.
- Retourne 1 si ce palindrome est carré et 0 sinon.

*void palindromicSquare(tab * palindromicSquares,champ_t * here, int input,champ_t *grid, int w, int h) ;*

- Cette procédure permet de placer la définition dans la grille et d'établir les différents nombres palindrome carré utilisable dans la grille.
- palindromicSquares est un vecteur qui va stocker les palindromes carré possible pour cette grille.
- here localise l'endroit où placer la fonction
- input est l'entrée de l'utilisateur
- grid nous donne accès à notre grille de travail.
- w correspond à la taille de la grille en largeur.
- h correspond à la taille de la grille en hauteur.

void forPalindromicSquare(tab * palindromicSquares,champ_t * here, int input,champ_t *grid, int w, int h)

- Celle-ci, nous va nous servir pour la prochaine fonction. En effet, à partir des premiers chiffres que nous avons autorisées, nous devons identifier quelle suite de chiffres est possible. Elles

diffèrent pour chaque palindrome carré. Celui-ci va donc mettre en place les suites de chiffres possibles à partir de la première case.
- palindromicSquares contient les différents nombres de la grille qui sont à la fois des palindromes et des carrés.
- here localise l'endroit où placer la fonction
- input est l'entrée de l'utilisateur
- grid nous donne accès à notre grille de travail.
- w correspond à la taille de la grille en largeur.
- h correspond à la taille de la grille en hauteur.

*void searchPalindromicSquare (void (*fct1)(int *, champ_t *, champ_t *,int, champ_t *,int, int), int * lastFct1, champ_t * hereFct1, int inputFct1, void (*fct2)(int *, champ_t *, champ_t *,int, champ_t *,int, int), int * lastFct2, champ_t * hereFct2, int inputFct2, void (*fct3)(int *, champ_t *, champ_t *,int, champ_t *,int, int), int * lastFct3, champ_t * hereFct3, int inputFct3, tab * palindromicSquares, champ_t * here, champ_t * ptrCaseNoire, int input, champ_t * grid,int w, int h)*

- Cette procédure est un début d'idée pour employer différentes fonctions à différents endroits de la grille. On utilise des pointeurs de fonctions qui changeront en fonction de l'entrée de l'utilisateur. Chaque fonction représente une partie d'où le premier chiffre doit correspondre à un des nombres palindrome carré possible. Nous lançons chaque fonction tant qu'elle ne coordonne pas avec la fonction pour les palindromes carrés.
- fct1, fct2 et fct3 sont les 3 fonctions choisis par l'utilisateur.
- Elles ont toutes les 3 un lastFct qui correspond aux derniers éléments testés pour chacun, un hereFct qui correspond à la localisation de chacune dans la grille et un inputFct qui contient l'entrée que l'utilisateur à entrée pour les 3.
- palindromicSquares contient les différents nombres de la grille qui sont à la fois des palindromes et des carrés.
- here localise l'endroit où placer la fonction
- input est l'entrée de l'utilisateur
- grid nous donne accès à notre grille de travail.
- w correspond à la taille de la grille en largeur.
- h correspond à la taille de la grille en hauteur.

3. Le main pour l'utilisateur

main.c : Cette fonction principale est ce qui permet à notre utilisateur d'utiliser le programme. Ainsi, il pourra avoir le choix entre différentes possibilités de problèmes. Malheureusement, nous n'avons pas terminé, nous voulions que l'utilisateur puisse à la fois choisir ses fonctions horizontales mais aussi vertical. La taille de la grille devra être possiblement étendue pour une utilisation d'une grille 5*5 ou même 6*6.

III. Traces d'utilisation

1. Exemples d'exécution

Cas 01 :

Case noire (2C) **1**) Chiffre consécutif _ 2) Nombre premier _ 3) Carré

```
                Bienvenue sur grille de nombre !!
        Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons travailler que sur une grille de taille 3*3) :
3
                  a      b         c
                ----------------------------
                1 |                    |
                2 |                    |
                3 |                    |
                ----------------------------
        Placez les définitions :
        Pour a :
        1. Carré et palindrome
1
        Placez les définitions :
        Pour b :
        1. Le produit des chiffres est 12
1
        Voulez-vous une case noire ? O ou N
O
        Entrez une lettre pour placer votre case noire (repérez-vous aux lettres aux dessus si besoin) :
        P.-S. Vous ne pouvez pas mettre de case noire sur les 2 colonnes précédentes !
c
        Entrez un chiffre pour placer votre case noire (repérez-vous aux chiffres à gauche si besoin) :
2
        Placez les définitions :
        Pour 1 :
        1. Suite de chiffre consécutifs
        2. Nombre premier
        3. Carré
1
        Placez les définitions :
        Pour 2 :
        1. Nombre premier
        2. Carré
1
        Placez les définitions :
        Pour 3 :
        1. Carré
1
                  a b c
                -----------
                1 | 1 2 3 |
                2 | 2 3 * |
                3 | 1 2 1 |
```

Cas 02 :

Case noire (2C) 1) carré _ 2) Chiffre consécutif_ 3) Nombre premier

```
                Bienvenue sur grille de nombre !!
        Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons
travailler que sur une grille de taille 3*3) :
3
                  a       b         c
                --------------------------
                1 |                      |
                2 |                      |
                3 |                      |
                --------------------------
        Placez les définitions :
        Pour a :
        1. Carré et palindrome
1
        Placez les définitions :
        Pour b :
        1. Le produit des chiffres est 12
1
        Voulez-vous une case noire ? O ou N
o
        Entrez une lettre pour placer votre case noire (repérez-vous aux lettres aux
dessus si besoin) :
        P.-S. Vous ne pouvez pas mettre de case noire sur les 2 colonnes précédentes !
c
        Entrez un chiffre pour placer votre case noire (repérez-vous aux chiffres à
gauche si besoin) :
2
        Placez les définitions :
        Pour 1 :
        1. Suite de chiffre consécutifs
        2. Nombre premier
        3. Carré
3
        Placez les définitions :
        Pour 2 :
        1. Suite de chiffre consécutifs
        2. Nombre premier
1
        Placez les définitions :
        Pour 3 :
        1. Nombre premier
1
                  a b c
                ----------
                1 | 1 2 1 |
                2 | 2 3 * |
                3 | 1 2 1 |
                ----------
```

Cas 03 :

Case noire (1C) 1) Chiffre consécutif _ 2) Nombre premier _ 3) Carré

Bienvenue sur grille de nombre !!

Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons travailler que sur une grille de taille 3*3) :

3

```
   a     b     c
-------------------------
1 |                     |
2 |                     |
3 |                     |
-------------------------
```

Placez les définitions :

Pour a :

1. Carré et palindrome

1

Placez les définitions :

Pour b :

1. Le produit des chiffres est 12

1

Voulez-vous une case noire ? O ou N

o

Entrez une lettre pour placer votre case noire (repérez-vous aux lettres aux dessus si besoin) :

P.-S. Vous ne pouvez pas mettre de case noire sur les 2 colonnes précédentes !

c

Entrez un chiffre pour placer votre case noire (repérez-vous aux chiffres à gauche si besoin) :

1

Placez les définitions :

Pour 1 :

1. Suite de chiffre consécutifs
2. Nombre premier
3. Carré

1

Placez les définitions :

Pour 2 :

1. Nombre premier
2. Carré

1

Placez les définitions :

Pour 3 :

1. Carré

1

```
   a b c
-----------
1 | 1 2 * |
2 | 2 1 1 |
3 | 1 6 1 |
-----------
```

Cas 04 :

(Pas de case noire) 1) Chiffre consécutif _ 2) Nombre premier _ 3) Carré

Bienvenue sur grille de nombre !!
Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons travailler que sur une grille de taille 3*3) :
3

```
   a       b          c
-------------------------
1 |                     |
2 |                     |
3 |                     |
-------------------------
```

Placez les définitions :
Pour a :
1. Carré et palindrome
1

Placez les définitions :
Pour b :
1. Le produit des chiffres est 12
1

Voulez-vous une case noire ? O ou N
n

Placez les définitions :
Pour 1 :
1. Suite de chiffre consécutifs
2. Nombre premier
3. Carré
1

Placez les définitions :
Pour 2 :
1. Nombre premier
2. Carré
1

Placez les définitions :
Pour 3 :
1. Carré
1

```
   a b c
-----------
1 | 1 2 3 |
2 | 2 1 1 |
3 | 1 6 1 |
-----------
```

2. Exemples d'erreur d'exécution

Cas 01 :

Case noire (2C) 1) Nombre premier_ 2) Chiffre consécutif _3) carré

```
                    Bienvenue sur grille de nombre !!
                Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons
travailler que sur une grille de taille 3*3) :
3
                      a       b       c
                    ------------------------
                    1 |                     |
                    2 |                     |
                    3 |                     |
                    ------------------------
                Placez les définitions :
                Pour a :
                1. Carré et palindrome
1
                Placez les définitions :
                Pour b :
                1. Le produit des chiffres est 12
1
                Voulez-vous une case noire ? O ou N
o
                Entrez une lettre pour placer votre case noire (repérez-vous aux lettres aux
dessus si besoin) :
                P.-S. Vous ne pouvez pas mettre de case noire sur les 2 colonnes précédentes !
c
                Entrez un chiffre pour placer votre case noire (repérez-vous aux chiffres à
gauche si besoin) :
2
                Placez les définitions :
                Pour 1 :
                1. Suite de chiffre consécutifs
                2. Nombre premier
                3. Carré
2
                Placez les définitions :
                Pour 2 :
                1. Suite de chiffre consécutifs
                2. Carré
1
                Placez les définitions :
                Pour 3 :
                1. Carré
1
Erreur une des cases est égale à 0
                      a b c
                    -----------
                    1 | 1 0 1 |
                    2 | 2 3 * |
                    3 | 1 2 1 |
                    -----------
```

Cas 02 :

Case noire (3C) 1) Carré _ 2) Nombre premier _ 3) Chiffres consécutif

Bienvenue sur grille de nombre !!
Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons travailler que sur une grille de taille 3*3) :
3

```
   a     b     c
-------------------------
1 |                     |
2 |                     |
3 |                     |
-------------------------
```

Placez les définitions :
Pour a :
1. Carré et palindrome
1
Placez les définitions :
Pour b :
1. Le produit des chiffres est 12
1
Voulez-vous une case noire ? O ou N
O
Entrez une lettre pour placer votre case noire (repérez-vous aux lettres aux dessus si besoin) :
P.-S. Vous ne pouvez pas mettre de case noire sur les 2 colonnes précédentes !
c

Entrez un chiffre pour placer votre case noire (repérez-vous aux chiffres à gauche si besoin) :
3
Placez les définitions :
Pour 1 :
1. Suite de chiffre consécutifs
2. Nombre premier
3. Carré
3
Placez les définitions :
Pour 2 :
1. Suite de chiffre consécutifs
2. Nombre premier
2
Placez les définitions :
Pour 3 :
1. Suite de chiffre consécutifs
1

```
  a b c
-----------
1 | 1 2 1 |
2 | 2 1 1 |
3 | 1 6 * |
```

Cas 03 : Taille de grille 6*6

Bienvenue sur grille de nombre !!

Entrez la taille souhaitée pour votre grille (pour l'instant, nous ne pouvons travailler que sur une grille de taille 3*3) :

6

```
  a     b       c       d       e       f
-----------------------------------------------------
1 |                                       |
2 |                                       |
3 |                                       |
4 |                                      |
5 |                                      |
6 |                                      |
-----------------------------------------------------
```

IV. Conclusion

Ce fut un défi, car ayant été perdues à un moment, nous avons pris pas mal de retard. Nous voulions absolument le remettre en avance donc d'un côté nous sommes déçu de ne pas avoir pu faire tout ce que nous avions en tête, mais de l'autre, on est réellement satisfaite d'avoir réussi à rattraper notre retard et rendu quelque chose qui se rapproche de notre objet final. Ce projet n'est pas achevé et nous verrons bien comment l'améliorer par la suite, mais pour l'instant, il représente tout le travail acharné qu'on a donné ce dernier mois.

V. Listing du programme

affichage.c :

```
#include "projet.h"

void printGrid(champ_t * grid, int w, int h)
{
	printf("\n\t\t\t    ");
	for (int i = 0; i < w; ++i)
	{
		printf("%c ", i+97);
	}

	printf("\n\t\t\t");
	for (int i = 0; i < w + 2 + 2 * w; i++)
		printf("-");
	printf("\n");

	for (int line = 0; line < h; ++line)
	{
		printf("\t\t\t%d", line+1);
		printf(" |");

		for (int col = 0; col < w; ++col)
		{
			if (grid[line * w + col].a == 10)
				printf("  ");
			else if (grid[line * w + col].a == 11)
				printf(" *");

			else
				printf(" %d", grid[line * w + col].a);
		}
		printf(" |\n");
	}

	printf("\t\t\t");
	for (int i = 0; i < w + 2 + 2 * w; i++)
		printf("-");
	printf("\n");
}

void affichamp(champ_t cell)
{
	printf("%u %u %u %u %u %u %u %u %u %u\n", cell.a, cell.b, cell.c, cell.d, cell.e, cell.f, cell.g, cell.h, cell.i, cell.j, cell.k);

}

void affTab(tab t)
{
	int *ptr = t.t;

	printf("{");

	for (Uint nb = t.nb; nb; nb--)
		printf(" %.d ", *ptr++);

	printf("}\n");
}
```

carre.c :

```
#include "projet.h"

int isSquare(int nb)
{
	int nbInt = (int) sqrt(nb);
	float nbTest = sqrt(nb);
```

```
	if(nbTest == nbInt)
		return 1;

	return 0;
}

void searchSquare(int * lastSquare, champ_t * here, champ_t * ptrCaseNoire,int input,champ_t * grid, int w, int h)
{
	champ_t *ptrGrid = (champ_t *) grid, *ptr = here;
	int nb = 0, digit, h2 = h;

	// ON PLACE LE POINTEUR A LA FIN
	// Si input est horizontal càd un chiffre
	if(input >= 0 && input <= 9)
	{
		ptrGrid = here + (w-1);
	}

	// Si input est vertical càd une lettre
	else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
	{
		ptrGrid += w * (h-1) + (input - 'a');
	}

	// On s'occupe de la case noire :
	if(ptrCaseNoire)
		toAvoidBlackCell(here, ptrCaseNoire, input, &ptrGrid, &w, &h2);

	// Si on a testé aucun nombre, donc que primes est vide
	if (!*lastSquare)
	{
		if(input >= 0 && input <= 9)
		{
			// On prend le nombre sur la colonne
			for (int exposant = w-1; exposant >= 0; --exposant)
			{
				if(ptr->a != 10 && ptr->a != 11)
				{
					nb += ptr++->a * pow(10, exposant);
				}

				else
					nb += 1 * pow(10, exposant);
			}
		}

		else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
		{
			// On prend le nombre sur la ligne
			for (int exposant = h2-1; exposant >= 0; --exposant)
			{
				if(ptr->a != 10 && ptr->a != 11)
					nb += ptr++->a * pow(10, exposant);
				else
					nb += 1 * pow(10, exposant);
			}
		}
	}

	// Sinon on cherche à partir du dernier nombre premier testé
	else
		nb = *lastSquare + 1;

	// On cherche un carré
	while(nb < 1 * pow(10,w) && !isSquare(nb))
		nb++;

	// Ajout dans le tableau des carrés testés
	*lastSquare = nb;

	// Ajout dans la grille
	while(nb > 0)
	{
		digit = nb % 10;
		nb = (nb-digit)/10;
```

```
                        // Dans un cas de remplissage horizontal
                        if(input >= 0 && input <= 9)
                        {
                                ptrGrid--->a = digit;

                        }
                        // Dans un cas de remplissage vertical
                        else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
                        {
                                ptrGrid->a = digit;
                                ptrGrid -= w;
                        }
        }
}
```

carre_Palindrome.c :

```
#include "projet.h"

int numberOfDigits(int nb)
{
        int res = 0;

        for (; nb > 0; ++res)
                nb /= 10;

        return res;
}

int doUpsideDown(int nb)
{
        int upsideDown = 0, chiffreNb;
        for (int i = numberOfDigits(nb) - 1, j = nb; i >= 0; --i)
        {
                chiffreNb = j % 10;
                j = (j-chiffreNb)/10;
                upsideDown = upsideDown + chiffreNb * pow(10, i);
        }
        return upsideDown;
}

int isPalindrome(int nb, int nbUpsideDown)
{
        if(nb == nbUpsideDown)
                return 1;
        else
                return 0;
}

int isPalindromicSquare(int nb)
{
        int nbUpsideDown = doUpsideDown(nb), nbInt; // Le nbInt va nous permettre de garder notre nombre
        float nbTest;

        if(isPalindrome(nb, nbUpsideDown))
        {
                nbInt = (int) sqrt(nb);
                nbTest = sqrt(nb);
                if(nbTest == nbInt)
                        return 1;
        }
        return 0;
}

void palindromicSquare(tab * palindromicSquares,champ_t * here, int input,champ_t *grid, int w, int h)
{
        int * ptrPalindromicSquare, firstDigit, * endPalindromicSquares = palindromicSquares->t + palindromicSquares->nb - 1;

        for (int i = 1 * pow(10,w - 1); i < 1 * pow(10,w); ++i)
        {
                doUpsideDown(i);
                if (isPalindromicSquare(i))
                {
```

```
                                        // Ajout dans les palindromes carrés testés
                                        palindromicSquares->nb += 1;
                                        palindromicSquares->t = (int *) realloc(palindromicSquares->t, palindromicSquares->nb * sizeof
*palindromicSquares->t);
                                        endPalindromicSquares = palindromicSquares->t + palindromicSquares->nb - 1;
                                        *endPalindromicSquares = i;
                                }
                        }

                        ptrPalindromicSquare = palindromicSquares->t;

                        // On va comparer la case du tableau avec les premiers chiffres des palindromes carrés
                        for (Uint nb = palindromicSquares->nb; nb; nb--, *ptrPalindromicSquare++)
                        {
                                firstDigit = (*ptrPalindromicSquare) / pow(10, (int) log10((*ptrPalindromicSquare)));
                                digitAllowed(here, firstDigit);
                        }
}

void forPalindromicSquare(tab * palindromicSquares,champ_t * here, int input,champ_t *grid, int w, int h)
{
        champ_t *ptrGrid = (champ_t *) grid;
        int firstDigit, digit, nbr, *ptrPalindromicSquare = palindromicSquares->t;

        if(isAllowed(here,here->a))
        {
                for (Uint nb = palindromicSquares->nb; nb; nb--, *ptrPalindromicSquare++)
                {
                        firstDigit = (*ptrPalindromicSquare) / pow(10, (int) log10((*ptrPalindromicSquare)));

                        if(firstDigit == here->a)
                        {
                                ptrGrid +=  w * (h-1) + (input - 'a');
                                nbr = (*ptrPalindromicSquare) - firstDigit * pow(10,w - 1);

                                while (nbr > 0)
                                {
                                        digit = nbr % 10;
                                        nbr = (nbr-digit)/10;
                                        digitAllowed(ptrGrid, digit);
                                        ptrGrid -= w;
                                }
                        }
                }
        }
}

void searchPalindromicSquare
(
        void (*fct1)(int *, champ_t *, champ_t *,int, champ_t *,int, int), int * lastFct1, champ_t * hereFct1, int inputFct1,
        void (*fct2)(int *, champ_t *, champ_t *,int, champ_t *,int, int), int * lastFct2, champ_t * hereFct2, int inputFct2,
        void (*fct3)(int *, champ_t *, champ_t *,int, champ_t *,int, int), int * lastFct3, champ_t * hereFct3, int inputFct3,
        tab * palindromicSquares, champ_t * here, champ_t * ptrCaseNoire, int input, champ_t * grid,int w, int h)
        {
                while(!isAllowed(here, here->a))
                {
                        (*fct1)(lastFct1, hereFct1, ptrCaseNoire, inputFct1, grid, w, h);
                }
                forPalindromicSquare(palindromicSquares,here, input,grid, w, h);
                here += w;


                while(!isAllowed(here, here->a))
                {
                        (*fct2)(lastFct1, hereFct2, ptrCaseNoire, inputFct2, grid, w, h);
                }
                here += w;

                while(!isAllowed(here, here->a))
                {
                        (*fct3)(lastFct3, hereFct3, ptrCaseNoire, inputFct3, grid, w, h);
                }

                here += w;
}
```

case_Noire.c :

```
#include "projet.h"

void initBlackCell(champ_t * here, champ_t ** ptrCaseNoire, int noireColonne, int noireLigne, champ_t * grid, int w, int h)
{
	here = inputGrid(noireLigne, grid, w, h) + (noireColonne - 'a');
	here->a = 11;
	(*ptrCaseNoire) = here;
}

void toAvoidBlackCell(champ_t * here, champ_t * ptrCaseNoire, int input,champ_t ** ptrGrid, int * w, int * h)
{
	champ_t * i = here;

	// On parcourt l'emplacement pour vérifier si il y a la case noire
	if(input >= 0 && input <= 9)
		while(i < *ptrGrid && ptrCaseNoire != i)
			i +=1;

	else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
		while(i < *ptrGrid && ptrCaseNoire != i)
			i += *w;

	if(ptrCaseNoire == i)
	{
		if(input >= 0 && input <= 9)
		{
			if(*ptrGrid == i)
			{

				*w -= 1;
				(*ptrGrid) -= 1;
			}

			else if(here < i)
			{
				*w = i - here;
				(*ptrGrid) = i - 1;
			}

			else if(here == i)
				*w -= 1;
		}

		// Dans ce cas là on va prendre une deuxième variable (h) pour ne pas modifier le h garde la vrai hauteur de la grille
et dont on va se resservir après
		else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
		{
			if(*ptrGrid == i)
			{
				*h -= 1;
				(*ptrGrid) -= *w;
			}

			else if(here < i)
			{
				(*ptrGrid) = i - *w;
				*h = ((*ptrGrid) - here)/ (*w) + 1;
			}

			else if(here == i)
				*h -= 1;
		}
	}
}
```

chiffres_Consecutifs.c :

```
#include "projet.h"

void consecutiveDigit(int * lastConsecutiveDigit,champ_t * here, champ_t * ptrCaseNoire,int input,champ_t *grid, int w, int h)
{
	int digit, endDigit, h2 = h;
	champ_t *ptrGrid = (champ_t *) grid;

	//Pointeur sur la derniere case
	champ_t * endHere = (champ_t *) here;

	// Si input est horizontal càd un chiffre
	if(input >= 0 && input <= 9)
		ptrGrid = here + (w-1);

	// Si input est vertical càd une lettre
	else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
	{
		ptrGrid += w * (h-1) + (input - 'a');
	}

	// On s'occupe de la case noire :
	if(ptrCaseNoire)
		toAvoidBlackCell(here, ptrCaseNoire, input, &ptrGrid, &w, &h2);


	// Si on a testé aucun nombre
	if (!*lastConsecutiveDigit)
	{
		// On prend le premier élément si il y a déjà un chiffre sinon 1
		if(here->a != 10 && here->a != 11)
		{
			digit = here->a;
			endDigit = digit + (w - 1);
		}

		else
		{
			digit = 1;
			endDigit = digit + (w - 1);
		}
	}

	// Sinon on cherche à partir du dernier nombre premier testé
	else
	{
		digit = *lastConsecutiveDigit + 1;

		// On prend le premier élément si il y a déjà un chiffre sinon 1
		if(here->a != 10 && here->a != 11)
			endDigit = digit + (w - 1);

		else
			endDigit = digit + (w - 1);
	}

	// Ajout du nombre comme dernier testé
	*lastConsecutiveDigit = digit;

	//pour remplir horizontalement
	if(input >= 0 && input <= 9)
	{
		endHere = here + (w-1);
		for (int i = 0; i < w; ++i)
		{
			endHere->a = endDigit--;
			endHere--;
		}
	}
```

```
	//pour remplir verticalement
	else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
	{
		endHere = here + (2 * h);
		for (int i = 0; i < h2; ++i)
		{
			endHere->a = endDigit--;
			endHere -= w;
		}
	}
}
```

chiffres_Interdits.c :

```
#include "projet.h"


void digitAllowed(champ_t * t, int chffr)
{
	switch(chffr)
	{
		case 0:
			t->b = 1;
			break;
		case 1:
			t->c = 1;
			break;
		case 2:
			t->d = 1;
			break;
		case 3:
			t->e = 1;
			break;
		case 4:
			t->f = 1;
			break;
		case 5:
			t->g = 1;
			break;
		case 6:
			t->h = 1;
			break;
		case 7:
			t->i = 1;
			break;
		case 8:
			t->j = 1;
			break;
		case 9:
			t->k = 1;
			break;
	}
}

int isAllowed(champ_t * t, int chffr)
{
	switch(chffr)
	{
		case 0:
			return t->b;
		case 1:
			return t->c;
		case 2:
			return t->d;
		case 3:
			return t->e;
		case 4:
			return t->f;
		case 5:
			return t->g;
		case 6:
			return t->h;
```

```
                case 7:
                        return t->i;
                case 8:
                        return t->j;
                case 9:
                        return t->k;
        }
        return 0;
}
```

initialisations.c :

```
#include "projet.h"

void initGrid(champ_t * grid, int w, int h)
{
        for(int line = 0; line < h; line++)
        {
                for(int col = 0; col < w; col++)
                {
                        grid[line * w + col].a = 10;
                }
        }
}


void initBits(champ_t * grid, int w, int h)
{
        for (int line = 0; line < h; ++line)
        {
                for (int col = 0; col < w; ++col)
                {
                        grid[line * w + col].b = 0;
                        grid[line * w + col].c = 0;
                        grid[line * w + col].d = 0;
                        grid[line * w + col].e = 0;
                        grid[line * w + col].f = 0;
                        grid[line * w + col].g = 0;
                        grid[line * w + col].h = 0;
                        grid[line * w + col].i = 0;
                        grid[line * w + col].j = 0;
                        grid[line * w + col].k = 0;
                }
        }
}

champ_t * inputGrid(int input,champ_t * grid, int w, int h)
{
        champ_t * addHere = NULL, *ptrGrid = grid;

        // Dans le cas où l'utilisateur entre un chiffre
        if(input > 0 && input <= 9)
        {
                if(input == 1)
                        addHere = ptrGrid;
                else
                        addHere = ptrGrid + w * (input - 1);
        }

        // Dans le cas où l'utilisateur entre une lettre
        else if(input >= 97)
        {
                addHere = ptrGrid + (input - 97);
        }

        else
                printf("Entrée incorrecte !\n");

        return addHere;
}
```

nombres_Premiers.c :

```
#include "projet.h"

int findPrimeNumber(int nb)
{
	int isPrime = 1, prime = nb + 1;

	for (int i = 2; i < prime && isPrime; ++i)
	{
		// nb n'est pas un nombre premier s'il a un diviseur autre que 1 ou lui même
		if (prime%i == 0)
			isPrime = 0;

		else
			isPrime = 1;
	}

	if(isPrime)
		return prime;
	return findPrimeNumber(nb+1);
}

void searchPrimeNumber(int * lastPrime,champ_t *here, champ_t * ptrCaseNoire, int input, champ_t * grid, int w, int h)
{
	champ_t *ptrGrid = (champ_t *) grid;
	int nb, digit, prime = 0, h2 = h;

	// Si input est horizontal càd un chiffre
	if(input >= 0 && input <= 9)
		ptrGrid = here + (w-1);

	// Si input est vertical càd une lettre
	else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
	{
		ptrGrid += w * (h-1) + (input - 'a');
	}

	// On s'occupe de la case noire :
	if(ptrCaseNoire)
		toAvoidBlackCell(here, ptrCaseNoire, input, &ptrGrid, &w, &h2);

	// On prend le premier élément si il y a déjà un chiffre
	if(input >= 0 && input <= 9)
	{
		if(here->a != 10 && here->a != 11)
			nb = here->a * pow(10, w-1);
		else
			nb = 1 * pow(10, w-1);
	}

	else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
	{
		if(here->a != 10 && here->a != 11)
			nb = here->a * pow(10, h2-1);
		else
			nb = 1 * pow(10, h2-1);
	}

	// Si on a testé aucun nombre
	if (!*lastPrime)
	{
		prime = findPrimeNumber(nb);
	}

	// Sinon on cherche à partir du dernier nombre premier testé
	else
		prime = findPrimeNumber(*lastPrime);

	// Ajout du nombre comme dernier testé
```

```
		*lastPrime = prime;



		// Ajout du nombre dans la grille
		while(prime > 0)
		{
			digit = prime % 10;
			prime = (prime-digit)/10;

			// Dans un cas de remplissage horizontal
			if(input >= 0 && input <= 9)
			{
				ptrGrid--->a = digit;
			}

			// Dans un cas de remplissage vertical
			else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
			{
				ptrGrid->a = digit;
				ptrGrid -= w;
			}
		}
}
```

produit.c :

```
#include "projet.h"


void produit(champ_t *here, int input,champ_t *grid, int w, int h)
{
		champ_t *ptrGrid = here;
		int ch1 = 1,ch2 = 1,ch3 = 1;

		// Si input est horizontal càd une chiffre
		if(input >= 0 && input <= 9)
		{
			ch1 = ptrGrid++->a;
			ch2 = ptrGrid++->a;
		}

		// Si input est vertical càd une lettre
		else if ((input >= 'a' && input <= 'z') || (input >= 'A' && input <= 'Z'))
		{
			ch1 = ptrGrid->a;
			ptrGrid += h;
			ch2 = ptrGrid->a;
			ptrGrid += h;
		}

		// Si l'une des cases est egal à 0 faut chercher d'autre valeurs
		if(ch1 * ch2 == 0)
		{
			printf(" Erreur une des cases est egale à 0\n");
		}

		else
		{
			ch3 = 12 / (ch1 * ch2);
			ptrGrid->a = ch3;
		}
}
```