#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#define N 3

#define TAILLE (N \* N)

typedef struct

{

int valeur;

bool t\_candidats[TAILLE + 1];

int nbCandidats;

}tCase;

typedef tCase tGrilleCand[TAILLE][TAILLE];

const char LIGNE\_SEPARATION[28] = " +-------+-------+-------+";

const char CARAC\_POINT[2] = ".";

const char CARAC\_PIPE[2] = "|";

const char ESPACE[2] = " ";

int chargerGrille(tGrilleCand g);

void affichageGrille(tGrilleCand grilleAffichage);

void initCand(tGrilleCand \*laCase, int lig, int col);

int fnNbCasesVides(tGrilleCand grilleSudoku);

void ajouterCandidat(tGrilleCand \*laCase, int lig, int col, int val);

void retirerCandidat(tGrilleCand \*laCase, int lig, int col, int val);

bool estCandidat(tGrilleCand laCase, int lig, int col, int val);

int fn\_nbCandidat(tGrilleCand laCase, int lig, int col);

void singletonNu(tGrilleCand \*grilleCand, int \*nbCasesVides);

void singletonCache();

void pairNue();

void paireCache();

int main()

{

tGrilleCand grilleCand;

// bool progression = true;

int nbCasesVides;

system("clear");

chargerGrille(grilleCand);

printf("Grille chargé avec succès !\n");

sleep(2);

nbCasesVides = fnNbCasesVides(grilleCand);

affichageGrille(grilleCand);

singletonNu(&grilleCand, &nbCasesVides);

for(int lig = 0; lig < TAILLE; lig++)

{

printf("\n");

for(int col = 0; col < TAILLE; col++)

{

printf("\nTableau [%d][%d] : ", lig, col);

for(int i = 0; i <= TAILLE; i++)

{

if(grilleCand[lig][col].t\_candidats[i] > 0)

{

printf("true ");

}

else

{

printf("false ");

}

}

printf(" Nombre de candidat : %d", grilleCand[lig][col].nbCandidats);

}

}

}

void singletonNu(tGrilleCand \*grilleCand, int \*nbCasesVides)

{

for(int lig = 0; lig < TAILLE; lig++)

{

for(int col = 0; col < TAILLE; col++)

{

/\*Initialisation de chaque candidat de chaque case à true( comme si ils étaient présent et ensuite mettre à

false ceux qui sont déjà présent dans la ligne, dans la colonne ou dans la case)\*/

initCand(grilleCand, lig, col);

if((\*grilleCand[lig][col]).valeur == 0)

{

//Parcours des lignes et met à false les cases du tableau de candidats quand la valeur est présente.

for (int j = 0; j < TAILLE; j++)

{

if((\*grilleCand[lig][j]).valeur != 0)

{

int valeurCase = (\*grilleCand[lig][j]).valeur;

retirerCandidat(grilleCand, lig, col, valeurCase);

}

}

//Parcours des colonnes et met à false les cases du tableau de candidats quand la valeur est présente.

for (int k = 0; k < TAILLE; k++)

{

if((\*grilleCand[k][col]).t\_candidats[0] != 0)

{

int valeurCase = (\*grilleCand[k][col]).valeur;

retirerCandidat(grilleCand, lig, col, valeurCase);

}

}

}

}

}

}

void ajouterCandidat(tGrilleCand \*laCase, int lig, int col, int val)

{

(\*laCase[lig][col]).t\_candidats[val] = true;

(\*laCase[lig][col]).nbCandidats++;

}

void retirerCandidat(tGrilleCand \*laCase, int lig, int col, int val)

{

(\*laCase[lig][col]).t\_candidats[val] = false;

(\*laCase[lig][col]).nbCandidats--;

}

bool estCandidat(tGrilleCand laCase, int lig, int col, int val)

{

bool estCandidat = false;

int indiceCandidat = 0;

while(indiceCandidat < TAILLE && !estCandidat)

{

if(laCase[lig][col].t\_candidats[indiceCandidat] == true)

{

estCandidat = true;

}

else

{

indiceCandidat++;

}

}

return estCandidat;

}

int fn\_nbCandidat(tGrilleCand laCase, int lig, int col)

{

int nbCandidats = 0;

nbCandidats = laCase[lig][col].nbCandidats;

return nbCandidats;

}

void singletonCache()

{

}

void pairNue()

{

}

void paireCache()

{

}

/\*\*

\* @fn chargerGrille(tgrille \*g)

\* @brief Procédure qui scanne l'entrée de l'utilisateur pour les lignes et colonnes et qui redemande si ce n'est pas un

\* entier entre 1 et 9 en signalant une erreur dans TOUS les autres cas.

\* @param g Grille du sudoku.

\*/

int chargerGrille(tGrilleCand g)

{

char nomFichier[30] = "GrilleA.sud"; //Variable qui va stocker le nom du fichier à charger dans la grille.

FILE \* f;

//printf("Nom du fichier ? (Grille[A-G].sud)\n");

//scanf("%s", nomFichier);

f = fopen(nomFichier, "rb");

if (f==NULL)

{

printf("\n Fichier %s n'existe pas.\n", nomFichier);

}

else

{

for(int lig = 0; lig < TAILLE; lig++)

{

for(int col = 0; col < TAILLE; col++)

{

fread(&g[lig][col].valeur, sizeof(int), 1, f);

}

}

}

fclose(f);

return fnNbCasesVides(g);

}

/\*\*

\* @fn affichageGrille(tgrille grilleAffichage)

\* @brief Procédure assez explicite dans le nom qui permet d'afficher la grille.

\* @param grilleAffichage Grille du sudoku suite au chargement fait précédement dans le main.

\*/

void affichageGrille(tGrilleCand grilleAffichage)

{

printf(" 1 2 3 4 5 6 7 8 9\n");

printf("%s\n", LIGNE\_SEPARATION);

for (int lig = 0; lig < TAILLE; lig++)

{

if (lig % N == 0 && lig != 0) //Si le parcour des lignes est un multiple de trois afficher la ligne séparatrice.

{

printf("%s\n", LIGNE\_SEPARATION);

}

printf("%d %s ", lig + 1, CARAC\_PIPE); // Affiche le premier pipe à chaques lignes.

for (int col = 0; col < TAILLE; col++)

{

if(grilleAffichage[lig][col].valeur == 0) //Change les zéros en point sinon affiche le nombre "normal".

{

printf("%s", CARAC\_POINT);

}

else

{

printf("%d", grilleAffichage[lig][col].valeur);

}

if (col % N == 2 && col != (TAILLE - 1)) //Si le parcours des colonnes est un multiple de trois afficher le pipe sinon faire simplement un espace

{

printf(" %s ", CARAC\_PIPE);

}

else

{

printf("%s", ESPACE);

}

}

printf("%s\n", CARAC\_PIPE); //Affiche le dernier pipe à la fin de chaque ligne

}

printf("%s\n", LIGNE\_SEPARATION);

}

int fnNbCasesVides(tGrilleCand grilleSudoku)

{

int nbCasesVides = 0;

for(int lig = 0; lig < TAILLE; lig++)

{

for(int col = 0; col < TAILLE; col++)

{

if (grilleSudoku[lig][col].valeur == 0)

{

nbCasesVides++;

}

}

}

return nbCasesVides;

}

void afficheStat(int nbCaseRemplis, int nbCaseARemplir, int nbCandidatInitial, int nbCandidatFinal)

{

}

void initCand(tGrilleCand \*laCase, int lig, int col)

{#include <stdlib.h>

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#define N 3

#define TAILLE (N \* N)

typedef struct

{

int valeur;

bool t\_candidats[TAILLE + 1];

int nbCandidats;

}tCase;

typedef tCase tGrilleCand[TAILLE][TAILLE];

const char LIGNE\_SEPARATION[28] = " +-------+-------+-------+";

const char CARAC\_POINT[2] = ".";

const char CARAC\_PIPE[2] = "|";

const char ESPACE[2] = " ";

int chargerGrille(tGrilleCand g);

void affichageGrille(tGrilleCand grilleAffichage);

void initCand(tGrilleCand \*laCase, int lig, int col);

int fnNbCasesVides(tGrilleCand grilleSudoku);

void ajouterCandidat(tGrilleCand \*laCase, int lig, int col, int val);

void retirerCandidat(tGrilleCand \*laCase, int lig, int col, int val);

bool estCandidat(tGrilleCand laCase, int lig, int col, int val);

int fn\_nbCandidat(tGrilleCand laCase, int lig, int col);

void singletonNu(tGrilleCand \*grilleCand, int \*nbCasesVides);

void singletonCache();

void pairNue();

void paireCache();

int main()

{

tGrilleCand grilleCand;

// bool progression = true;

int nbCasesVides;

system("clear");

chargerGrille(grilleCand);

printf("Grille chargé avec succès !\n");

sleep(2);

nbCasesVides = fnNbCasesVides(grilleCand);

affichageGrille(grilleCand);

singletonNu(&grilleCand, &nbCasesVides);

for(int lig = 0; lig < TAILLE; lig++)

{

printf("\n");

for(int col = 0; col < TAILLE; col++)

{

printf("\nTableau [%d][%d] : ", lig, col);

for(int i = 0; i <= TAILLE; i++)

{

if(grilleCand[lig][col].t\_candidats[i] > 0)

{

printf("true ");

}

else

{

printf("false ");

}

}

printf(" Nombre de candidat : %d", grilleCand[lig][col].nbCandidats);

}

}

}

void singletonNu(tGrilleCand \*grilleCand, int \*nbCasesVides)

{

for(int lig = 0; lig < TAILLE; lig++)

{

for(int col = 0; col < TAILLE; col++)

{

/\*Initialisation de chaque candidat de chaque case à true( comme si ils étaient présent et ensuite mettre à

false ceux qui sont déjà présent dans la ligne, dans la colonne ou dans la case)\*/

initCand(grilleCand, lig, col);

if((\*grilleCand[lig][col]).valeur == 0)

{

//Parcours des lignes et met à false les cases du tableau de candidats quand la valeur est présente.

for (int j = 0; j < TAILLE; j++)

{

if((\*grilleCand[lig][j]).valeur != 0)

{

int valeurCase = (\*grilleCand[lig][j]).valeur;

retirerCandidat(grilleCand, lig, col, valeurCase);

}

}

//Parcours des colonnes et met à false les cases du tableau de candidats quand la valeur est présente.

for (int k = 0; k < TAILLE; k++)

{

if((\*grilleCand[k][col]).t\_candidats[0] != 0)

{

int valeurCase = (\*grilleCand[k][col]).valeur;

retirerCandidat(grilleCand, lig, col, valeurCase);

}

}

}

}

}

}

void ajouterCandidat(tGrilleCand \*laCase, int lig, int col, int val)

{

(\*laCase[lig][col]).t\_candidats[val] = true;

(\*laCase[lig][col]).nbCandidats++;

}

void retirerCandidat(tGrilleCand \*laCase, int lig, int col, int val)

{

(\*laCase[lig][col]).t\_candidats[val] = false;

(\*laCase[lig][col]).nbCandidats--;

}

bool estCandidat(tGrilleCand laCase, int lig, int col, int val)

{

bool estCandidat = false;

int indiceCandidat = 0;

while(indiceCandidat < TAILLE && !estCandidat)

{

if(laCase[lig][col].t\_candidats[indiceCandidat] == true)

{

estCandidat = true;

}

else

{

indiceCandidat++;

}

}

return estCandidat;

}

int fn\_nbCandidat(tGrilleCand laCase, int lig, int col)

{

int nbCandidats = 0;

nbCandidats = laCase[lig][col].nbCandidats;

return nbCandidats;

}

void singletonCache()

{

}

void pairNue()

{

}

void paireCache()

{

}

/\*\*

\* @fn chargerGrille(tgrille \*g)

\* @brief Procédure qui scanne l'entrée de l'utilisateur pour les lignes et colonnes et qui redemande si ce n'est pas un

\* entier entre 1 et 9 en signalant une erreur dans TOUS les autres cas.

\* @param g Grille du sudoku.

\*/

int chargerGrille(tGrilleCand g)

{

char nomFichier[30] = "GrilleA.sud"; //Variable qui va stocker le nom du fichier à charger dans la grille.

FILE \* f;

//printf("Nom du fichier ? (Grille[A-G].sud)\n");

//scanf("%s", nomFichier);

f = fopen(nomFichier, "rb");

if (f==NULL)

{

printf("\n Fichier %s n'existe pas.\n", nomFichier);

}

else

{

for(int lig = 0; lig < TAILLE; lig++)

{

for(int col = 0; col < TAILLE; col++)

{

fread(&g[lig][col].valeur, sizeof(int), 1, f);

}

}

}

fclose(f);

return fnNbCasesVides(g);

}

/\*\*

\* @fn affichageGrille(tgrille grilleAffichage)

\* @brief Procédure assez explicite dans le nom qui permet d'afficher la grille.

\* @param grilleAffichage Grille du sudoku suite au chargement fait précédement dans le main.

\*/

void affichageGrille(tGrilleCand grilleAffichage)

{

printf(" 1 2 3 4 5 6 7 8 9\n");

printf("%s\n", LIGNE\_SEPARATION);

for (int lig = 0; lig < TAILLE; lig++)

{

if (lig % N == 0 && lig != 0) //Si le parcour des lignes est un multiple de trois afficher la ligne séparatrice.

{

printf("%s\n", LIGNE\_SEPARATION);

}

printf("%d %s ", lig + 1, CARAC\_PIPE); // Affiche le premier pipe à chaques lignes.

for (int col = 0; col < TAILLE; col++)

{

if(grilleAffichage[lig][col].valeur == 0) //Change les zéros en point sinon affiche le nombre "normal".

{

printf("%s", CARAC\_POINT);

}

else

{

printf("%d", grilleAffichage[lig][col].valeur);

}

if (col % N == 2 && col != (TAILLE - 1)) //Si le parcours des colonnes est un multiple de trois afficher le pipe sinon faire simplement un espace

{

printf(" %s ", CARAC\_PIPE);

}

else

{

printf("%s", ESPACE);

}

}

printf("%s\n", CARAC\_PIPE); //Affiche le dernier pipe à la fin de chaque ligne

}

printf("%s\n", LIGNE\_SEPARATION);

}

int fnNbCasesVides(tGrilleCand grilleSudoku)

{

int nbCasesVides = 0;

for(int lig = 0; lig < TAILLE; lig++)

{

for(int col = 0; col < TAILLE; col++)

{

if (grilleSudoku[lig][col].valeur == 0)

{

nbCasesVides++;

}

}

}

return nbCasesVides;

}

void afficheStat(int nbCaseRemplis, int nbCaseARemplir, int nbCandidatInitial, int nbCandidatFinal)

{

}

void initCand(tGrilleCand \*laCase, int lig, int col)

{

for (int i = 0; i <= TAILLE; i++)

{

(\*laCase[lig][col]).t\_candidats[i] = true;

(\*laCase[lig][col]).nbCandidats = TAILLE;

}

(\*laCase[lig][col]).t\_candidats[0] = false;

}

for (int i = 0; i <= TAILLE; i++)

{

(\*laCase[lig][col]).t\_candidats[i] = true;

(\*laCase[lig][col]).nbCandidats = TAILLE;

}

(\*laCase[lig][col]).t\_candidats[0] = false;

}