Nom :	
Prénom :	
No. groupe :	
No. carte :	

Éléments de programmation 2– 1I002

Partiel du 18 mars 2019
1h30
Aucun document n'est autorisé.

Les calculatrices, baladeurs et autres appareils électroniques sont interdits. Les téléphones mobiles doivent être éteints et rangés dans les sacs.

Toutes les questions sont indépendantes. Pour les questions à choix multiples à 1 point, vous obtenez 1 point si vous avez coché toutes les cases correspondant à des réponses justes et seulement celles-ci. Pour les questions acceptant plusieurs réponses toute réponse erronée (case juste non cochée ou case fausse cochée) est pénalisée. La note minimale à une question est 0. Le barème sur 22 points (12 questions) n'a qu'une valeur indicative.

ATTENTION : lisez le sujet dans son intégralité avant de commencer. Certaines questions sont à réponse libre (lecture ou écriture de code). Elles peuvent donc nécessiter plus de temps de réflexion que les questions à choix multiples.

Il ne vous est pas demandé de vérifier qu'un `malloc` a bien alloué la mémoire.

Questions de cours

Question 1 (2 points)

Soit les déclarations suivantes :

```
int v1;
int v2 = 0;
char v3 = 'A';
char v4 = "B";
float v5 = 3.14;
```

Cochez la ou les affirmations exactes en considérant que les déclarations suivantes sont réalisées à l'intérieur d'une fonction (variables locales).

- ■ La variable `v1` est correctement déclarée.
- □ La variable `v1` n'est pas correctement déclarée.
- ■ La variable `v1` est de type entier signé.
- □ La variable `v1` est de type entier non signé.
- □ La variable `v1` est initialisée par défaut à 0.
- ■ La variable `v1` n'est pas initialisée.
- ■ La variable `v2` est correctement déclarée.
- □ La variable `v2` n'est pas correctement déclarée.
- □ La variable `v2` n'est pas initialisée.
- ■ La variable `v2` est initialisée à 0.
- ■ La variable `v3` est correctement déclarée.
- □ La variable `v3` n'est pas correctement déclarée.
- ■ La variable `v3` est initialisée avec le code ASCII de la lettre A.
- □ La variable `v4` est correctement déclarée.
- ■ La variable `v4` n'est pas correctement déclarée.
- □ La variable `v4` correspond à une chaîne de caractères.
- ■ La variable `v5` est correctement déclarée.
- □ La variable `v5` n'est pas correctement déclarée.
- ■ La variable `v5` correspond à un flottant et est initialisée avec une valeur approchée de π.

Question 2 (2 points)

Soit la séquence de code suivante :

```
int tab1[3];
int tab2[] = {8,9};
int *p1 = tab1;
int *p2 = tab2;

tab1[1] = *p2;
*p1 = 7;
tab1[2] = 5;
```

Cochez la ou les affirmations exactes.

- ■ La variable `tab1` est correctement déclarée.
- □ La variable `tab1` n'est pas correctement déclarée.
- ■ La variable `tab1` correspond à un tableau de 3 entiers.
- ■ La variable `tab2` est correctement déclarée.
- □ La variable `tab2` n'est pas correctement déclarée.
- ■ La variable `tab2` correspond à un tableau de 2 entiers.
- □ Les variables `p1` et `p2` sont de type entier.
- ■ Les variables `p1` et `p2` sont de type pointeur sur entier.
- □ Après exécution des instructions le tableau `tab1` contient dans l'ordre des indices croissants les valeurs 7, 8, 9.
- ■ Après exécution des instructions le tableau `tab1` contient dans l'ordre des indices croissants les valeurs 7, 8, 5.
- □ Après exécution des instructions le tableau `tab1` contient dans l'ordre des indices croissants les valeurs 6, 7, 8.

Question 3 (1 point)

Soit la séquence de code suivante :

```
if (a <4) {
   if (b >= 5) {
      res = 2;
      }
   else {
      res = 3;
      }
else {
   res = 4;
   }
```

Cochez la ou les affirmations exactes.

- ■ Si $a = 3$ et $b = 6$ alors res reçoit la valeur 2.
- □ Si $a = 3$ et $b = 5$ alors res reçoit la valeur 3.
- ■ Si $a = 5$ et $b = 5$ alors res reçoit la valeur 4.
- □ Si $a = 5$ et $b = 5$ alors res reçoit la valeur 2.
- □ Si $a = 1$ et $b = 4$ alors res reçoit la valeur 2.
- ■ Si $a = 2$ et $b = 3$ alors res reçoit la valeur 3.
- □ Si $a = 6$ et $b = 4$ alors res reçoit la valeur 2.

Boucles et fonctions

Question 4 (2 points)

On veut écrire une fonction qui retourne 1 si le nombre fourni en argument est premier (seulement divisible par lui même et 1) et 0 sinon.

```
int EstPremier(int n)
   {
   int i;

   Instruction 1
      if ((n % i) == 0) {
         return 0;
         }
      }
   return 1;
   }
```

On rappelle que l'opérateur % correspond au reste de la division entière (modulo), l'instruction 1 correspond à une boucle. Quelle(s) instruction(s) de boucle permet(tent) d'obtenir le bon résultat.

- □ `for (i = 1; i < n; i++){`
- □ `for (i = 2; i <= n; i++){`
- ■ `for (i = 2; i < n; i++){`
- ■ `for (i = 2; i <= n/2; i++){`
- □ `for (i = 3; i <= n/2; i++){`
- □ `for (i = 3; i < n/2; i++){`

Question 5 (2 points)

Soit la fonction suivante :

```
int Fn(char *ch, char c)
   {
   int cpt = 0;
   while (*ch != '\0') {
      if (*ch == c) {
         cpt++;
         }
      ch++;
      }

   return cpt;
   }
```

Cochez la ou les affirmations exactes.

- ☐ La fonction retourne 4 quand elle est appelée comme suit : `Fn("abcd", 'a')`
- ■ La fonction retourne 1 quand elle est appelée comme suit : `Fn("abcd", 'a')`
- ■ La fonction retourne 0 quand elle est appelée comme suit : `Fn("abcd", 'f')`
- ■ La fonction retourne 0 quand elle est appelée comme suit : `Fn("", 'x')`
- ☐ La fonction retourne 1 quand elle est appelée comme suit : `Fn("", '\0')`
- ■ La fonction retourne 0 quand elle est appelée comme suit : `Fn("", '\0')`
- ☐ La fonction retourne la longueur de la chaîne `ch`.
- ■ La fonction retourne le nombre d'occurrences du caractère `c` dans la chaîne `ch`.

On veut maintenant écrire une forme récursive de la même fonction.

```
int FnRec(char *ch, char c)
   {
   if (/*Condition 1*/) {
      return 0;
      }
   if (/*Condition 2*/) {
      return FnRec( ch+1, c) + 1;
      }
   return FnRec( ch+1, c);
   }
```

Question 6 (1 point)

Cochez la ou les instruction(s) correspondant à `Condition 1`.

- ☐ `*ch`
- ☐ `ch == 0`
- ☐ `ch != 0`
- ☐ `ch[c] != 0`
- ☐ `*ch == c`
- ■ `*ch == '\0'`

Question 7 (1 point)

Cochez la ou les instruction(s) correspondant à `Condition 2`.

- ☐ `*ch == 'c'`
- ☐ `ch == c`
- ■ `*ch == c`
- ☐ `*ch != c`
- ☐ `ch[0] != c`
- ■ `ch[0] == c`

Question 8 (1 point)

On veut écrire une fonction retournant un tableau de `nb` entiers tous initialisés à 0, son prototype est :

```
int *CreerTableau(int nb)
```

Cochez la ou les implantations ne provocant ni warning ni erreur à la compilation ou erreur à l'exécution :

☐
```
int *CreerTableau(int nb) {
int tab[nb]; int i;

for (i=0; i< nb; i++) {
   tab[i] = 0;}

return tab; }
```

☐
```
int *CreerTableau(int nb) {
int *tab; int i;

tab = malloc(nb);
for (i=0; i< nb; i++) {
   tab[i] = 0;}

return tab; }
```

■
```
int *CreerTableau(int nb) {
int *tab; int i;

tab = malloc(nb * sizeof(int));
for (i=0; i< nb; i++) {
   tab[i] = 0;}

return tab; }
```

☐
```
int *CreerTableau(int nb) {
int *tab = malloc(nb * sizeof(int));

while (nb >0) {
   nb --;
   *tab = 0;
   tab++;}

return tab; }
```

■
```
int *CreerTableau(int nb) {
int *tab, *tab2;

tab = malloc(nb * sizeof(int));
tab2 = tab;
while (nb > 0) {
   nb --;
   *tab = 0;
   tab++;}

return tab2; }
```

■
```
int *CreerTableau(int nb) {
int *tab = malloc(nb * sizeof(int));

tab = tab + nb;
while (nb > 0) {
   nb --;
   tab --;
   *tab = 0; }

return tab; }
```

□
```
int *CreerTableau(int nb) {
int *tab = malloc(nb * sizeof(int));

tab = tab + nb;
while (nb > 0) {
   nb --;
   *tab = 0;
   tab --; }

return tab; }
```

Problème

L'objectif du jeu Sudoku est de compléter un tableau de 9×9 cases avec des valeurs comprises dans l'intervalle $[1, 9]$ de façon à ce que :

— Aucune ligne ne contienne 2 fois la même valeur ;
— Aucune colonne ne contienne 2 fois la même valeur ;
— Aucun carré (de 3×3) ne contienne 2 fois la même valeur.

Voici un exemple de Sudoku avant et après résolution.

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Sudoku non résolu

4	2	*6*	5	*7*	1	*3*	9	*8*
8	*5*	*7*	2	*9*	3	*1*	*4*	6
1	3	*9*	*4*	6	*8*	*2*	7	*5*
9	*7*	1	*3*	*8*	*5*	6	*2*	*4*
5	4	*3*	*7*	*2*	*6*	*8*	1	9
6	*8*	2	*1*	*4*	*9*	7	*5*	*3*
7	9	*4*	*6*	3	*2*	*5*	8	*1*
2	*6*	*5*	8	*1*	4	*9*	*3*	7
3	1	*8*	9	*5*	7	*4*	6	*2*

Sudoku résolu

L'objectif de ce devoir est d'écrire un programme apte à résoudre de façon automatique un Sudoku.

représentation informatique

Pour représenter le Sudoku nous allons utiliser un tableau dynamique de 81 (9×9) entiers, **la valeur 0 signifie que la case est vide**.

Pour cet exercice nous n'allons pas nous soucier de la façon dont est implémentée la matrice correspondant au Sudoku. Il sera possible de lire la valeur d'une case grâce à la fonction `int ValSudoku (int i, int j)` qui retourne la valeur de la case d'indice (i, j) et d'en affecter la valeur grâce à `void SetValSudoku(int i, int j, int val)`. i représente l'indice de ligne et j celui de la colonne. Par souci de simplification le programme ne manipulant qu'un seul Sudoku à la fois, celui-ci est déclaré dans une variable globale accessible de toutes les fonctions. Toutes les questions de ce problème sont indépendantes, vous pouvez les traiter dans l'ordre qui vous convient.

Question 9 (2 points)

On veut pouvoir afficher dans le terminal un Sudoku, la fonction `void print_sudoku()` doit produire l'affichage suivant pour le Sudoku présenté en exemple à la page précédente :

```
+=+=+=+=+=+=+=+=+=+
|   2   |5    1|   9   |
+-+-+-+-+-+-+-+-+-+
|8      |2    3|      6|
+-+-+-+-+-+-+-+-+-+
|   3   |   6   |   7   |
+=+=+=+=+=+=+=+=+=+
|      1|       |6      |
+-+-+-+-+-+-+-+-+-+
|5 4    |       |   1 9|
+-+-+-+-+-+-+-+-+-+
|      2|       |7      |
```

```
+=+=+=+=+=+=+=+=+=+
|   9   |   3   |   8   |
+-+-+-+-+-+-+-+-+-+-+
|2      |8    4 |      7|
+-+-+-+-+-+-+-+-+-+-+
|   1   |9    7 |   6   |
+=+=+=+=+=+=+=+=+=+=+
```

Nous avons écrit le squelette de cette fonction, à vous de le compléter :

```
void print_sudoku()
   {
   int i, j;
   printf("+=+=+=+=+=+=+=+=+=+=+\n");
   for(i=0; i<9; i++) {
      for(j=0; j<9; j++) {
         int val = ValSudoku(i, j);
         char cval;

         if (val == 0) { cval = '␣'; }
         else { cval = '0' + val; }

         /* Zone a completer */
         }

      if (i%3 == 2)
         printf("|\n+=+=+=+=+=+=+=+=+=+=+\n");
      else
         printf("|\n+-+-+-+-+-+-+-+-+-+-+\n");
      }
   }
```

Par quelles instructions compléter la fonction afin d'obtenir rigoureusement l'affichage souhaité, cochez la ou les case(s) correspondante(s) :

☐
```
if ((j%3) == 0) { printf("|%c", val);}
else { printf("␣%c", val);}
```

☐
```
if ((j%3) == 0) { printf("|%d", val);}
else { printf("␣%d", val);}
```

■
```
if ((j%3) == 0) { printf("|%c", cval);}
else { printf("␣%c", cval);}
```

☐
```
if ((j%3) == 0) { printf("|%d", cval);}
else { printf("␣%d", cval);}
```

Question 10 (2 points)

Pour résoudre le Sudoku il va falloir déterminer l'ensemble des valeurs présentes sur une ligne donnée, vous allez devoir pour cela écrire la fonction `int *ValsLigne(int i)`. Cette fonction doit retourner un tableau d'entiers correspondant à des booléens (0 dans la case d'indice $j-1$ si la valeur j n'est pas présente dans la ligne i et 1 sinon).

Solution:

```
int ValsLigne(int i)
   {
   int j;
   int *Tvals = malloc(9*sizeof(int));
   for (j=0; j<9; j++) { Tvals[j] = 0; }
   for (j=0; j<9; j++) {
      int val = ValSudoku(i,j);
      if (val) { Tvals[val -1] = 1; }}

   return Tvals;
   }
```

Les deux fonctions suivantes permettent de compléter le tableau crée par `ValsLigne` en prenant en compte les valeurs présentes dans une colonne puis celles présentes dans un carré de 3×3.

```
int *ValsColonne(int *Tvals, int j)
int *ValsCarre(int *Tvals, int i, int j)
```

Question 11 (2 points)

Une fois le tableau complété pour les ligne, colonne et carré par les fonctions précédemment présentées, la fonction `int NbVals(int *Tvals)` détermine le nombre de valeurs déjà présentes. Elle retourne une valeur comprise entre 0 et 9, (9 signifie que toutes les valeurs sont déjà présentes). Écrivez cette fonction.

Solution:

```
int NbVals(int *Tvals)
   {
   int i;
   int nb = 0;

   for (i=0; i<9; i++)
      if (Tvals[i]) nb++;

   return nb;
   }
```

Question 12 (4 points)

Le principe général de l'algorithme de résolution du Sudoku est récursif : le Sudoku est résolu si il n'y a plus de case vide. Si ce n'est pas le cas on place une nouvelle valeur possible dans une case vide puis on réitère l'algorithme jusqu'à ce qu'il n'y ait aucune case de vide. Si on n'y parvient pas on essaie avec une autre valeur.

Pour limiter le nombre de solutions à explorer au lieu de placer une valeur dans n'importe quelle case vide, nous allons sélectionner la case vide pour laquelle le nombre de valeurs possibles est le plus réduit. Nous allons pour cela déterminer la case vide pour laquelle le nombre de valeurs déjà présentes sur les ligne, colonne et carré est le plus grand.

La fonction `int SolveSudokuCini()` va tenter de résoudre le Sudoku, elle retourne 1 si elle y parvient et 0 sinon. Complétez cette fonction.

```
int SolveSudokuCini()
   {
   int *MaxTvals;/*Tab des vals deja presentes pr la case optimale*/
   int NbMax = 0;/*Nb de vals deja presentes pour la case optimale*/
   int ImaxTvals, JmaxTvals;/*Indices de la case optimale*/
   int i, j;
   int SudokuResolu = 1;

/* Recherche de la case vide avec le plus de valeurs deja presentes
   sur les ligne, colonne et carre*/
   for (i=0; i<9; i++) {
      for (j=0; j<9; j++) {
         if (ValSudoku(i, j) == 0)
            {
```

```
            int *Tvals;
            int Nb;

            /*A completer*/
            SudokuResolu = _____;

            Tvals = ValsLigne(i);
            Tvals = ValsColonne(Tvals, j);
            Tvals = ValsCarre(Tvals, i, j);
            Nb = NbVals(Tvals);
            if (Nb > NbMax)
               {
               /*A completer*/
               _____________________________________________
               _____________________________________________
               _____________________________________________
               _____________________________________________
               _____________________________________________
               _____________________________________________
               _____________________________________________
               _____________________________________________
               }
            else
               free(Tvals);
            }
         }
      }
   /*Cas ou il n y a pas plus de case vide*/
   if (SudokuResolu)
      {
      /*A completer*/
      ___________________________________________
      ___________________________________________
      }

/* Pour la case vide identifee precedemment, y placer des valeurs
   possibles et tenter de resoudre la suite du Sudoku*/

   for (i=0; i<9; i++)
      if (MaxTvals[i] ==0)
         {
         /*A completer*/
         ___________________________________________
         ___________________________________________
         ___________________________________________
         ___________________________________________
         ___________________________________________
         ___________________________________________
```

```
_____
_____
_____
_____
_____
_____
_____
}
```

```
/* Le Sudoku n a pas de solutions !*/
   free(MaxTvals);
   return 0;
   }
```

Vous prendrez soin de libérer avec `free` toute la mémoire allouée avec `malloc` en particulier celle allouée dans la fonction `ValsLigne`.

Solution:

```
int SolveSudokuCini()
   {
   int *MaxTvals;
   int NbMax = 0;
   int ImaxTvals, JmaxTvals;
   int i, j;
   int SudokuResolu = 1;

   for (i=0; i<9; i++)
      for (j=0; j<9; j++)
         if (ValSudoku(i, j) == 0)
            {
            int *Tvals;
            int Nb;

            SudokuResolu = 0;
            Tvals = ValsLigne(i);
            Tvals = ValsColonne(Tvals, j);
            Tvals = ValsCarre(Tvals, i, j);
            Nb = NbVals(Tvals);
            if (Nb > NbMax)
               {
               NbMax = Nb;
               ImaxTvals = i;
               JmaxTvals = j;
               MaxTvals = Tvals;
               }
            else
               free(Tvals);
            }
   if (SudokuResolu) return 1;
```

```
for (i=0; i<9; i++)
   if (MaxTvals[i] ==0)
      {
      SetValSudoku(ImaxTvals, JmaxTvals, i+1);
      if (SolveSudokuCini())
         {
         free(MaxTvals);
         return 1;
         }
      SetValSudoku(ImaxTvals, JmaxTvals, 0);
      }

free(MaxTvals);
return 0;
}
```