Université Hassan 1er

Faculté des Sciences et Techniques de Settat

PROGRAMMATION EN «C» ET STRUCTURES DE DONNÉES TRAVAUX DIRIGÉS + SOLUTIONS

Professeur Laachfoubi Nabil

Département des Mathématiques et Informatique

Sommaire

SERIE 1 – Les Tests Logiques, Les Procédures et Les Fonctions

1. Soient les procédures suivantes :

Algorithme Procedure P1( ) **Var** A, B, C **: Entier ;** **Debut** **Lire (**A , B**)** ; C ← A ; A ← B ; B ← C ; **Afficher (**A , B**)** ; **Fin**	**Algorithme Procedure P2( )** **Var** A, B **: Entier ;** **Debut** **Lire (**A , B**)** ; A ← A + B ; B ← A - B ; A ← A - B ; **Afficher (**A , B**)** ; **Fin**

   a) Remplissez les cellules grisées sur les tableaux de valeurs ci-dessous.
   b) Quel rôle joue chacune des procédures ?
   c) Laquelle de ces procédures est la plus rapide ?

Procédure P1				
Entrée			**Sortie**	
A	**B**		**A**	**B**
0	2	→		
-1	1			
4	2			

Procédure P2				
Entrée			**Sortie**	
A	**B**		**A**	**B**
0	2	→		
-1	1			
4	2			

2. Ecrire une procédure permettant d'afficher la valeur max entre deux entiers qu'un utilisateur saisit au clavier.

3. Ecrire une fonction permettant de retourner la valeur max entre deux entiers qu'un utilisateur saisit au clavier.

4. Ecrire une procédure prenant en paramètre deux variables de type entier et permettant d'afficher la valeur max entre ces deux variables.

5. Ecrire une fonction prenant en paramètre deux variables de type entier et permettant de retourner la valeur max entre ces deux variables.

6. Refaire les exercices 2, 3, 4 et 5 mais cette fois-ci en prenant en compte trois valeurs entières.

SERIE 2 – Les Boucles et La Récursivité

7. Soit la fonction f(n) = n! avec (0! = 1 et n! = n*(n-1)! Pour tout n > 0)
   a) Ecrire une procédure sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et d'afficher la valeur de f(n).
   b) Ecrire une procédure avec comme paramètre la variable n, permettant d'afficher la valeur de f(n).
   c) Ecrire une fonction sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et de retourner la valeur de f(n).
   d) Ecrire une fonction avec comme paramètre la variable n, permettant de retourner la valeur de f(n).
   e) Ecrire une procédure avec comme paramètres les variables n et fact, permettant de calculer la valeur de f(n) et stocker le résultat dans l'adresse de fact.
   f) Ecrire une procédure, permettant de trouver la plus petite valeur de n ainsi que celle de f(n) pour laquelle f(n) >= 121.
   g) Ecrire une fonction récursive avec comme paramètre la variable n et permettant de retourner la valeur de f(n).

8. Soit la suite U(n) = 3*U(n - 1) + 2 pour tout n > 0 avec U(0) = 1
   a. Ecrire une procédure sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et d'afficher la valeur de U(n).
   b. Ecrire une procédure avec comme paramètre la variable n, permettant d'afficher la valeur de U(n).
   c. Ecrire une fonction sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et de retourner la valeur de U(n).
   d. Ecrire une fonction avec comme paramètre la variable n, permettant de retourner la valeur de U(n).
   e. Ecrire une procédure, permettant de trouver la plus petite valeur de n ainsi que celle de U(n) pour laquelle U(n) >= 121.
   f. Ecrire une fonction récursive avec comme paramètre la variable n et permettant de retourner la valeur de U(n).
   g. Ecrire une procédure récursive avec comme paramètres la variable n et une autre variable passée par adresse et permettant de retourner la valeur de U(n).

9. Soit la série suivante :

$$S(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \quad avec\ n\ entier > 0$$

   a. Ecrire une procédure sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et d'afficher la valeur de S(n).
   b. Ecrire une procédure avec comme paramètre la variable n, permettant d'afficher la valeur de S(n).
   c. Ecrire une fonction sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et de retourner la valeur de S(n).
   d. Ecrire une fonction avec comme paramètre la variable n, permettant de retourner la valeur de S(n).
   e. Ecrire une procédure, permettant de trouver la plus petite valeur de n ainsi que celle de S(n) pour laquelle S(n) >= 3.
   f. Ecrire une fonction récursive avec comme paramètre la variable n et permettant de retourner la valeur de S(n).
   g. Ecrire une procédure récursive avec comme paramètres la variable n et une autre variable passée par adresse et permettant de retourner la valeur de S(n).

10. Ecrire une procédure permettant de résoudre l'équation de second degré $f(x) = a.x^2 + b.x + c = 0$

11. Ecrire une procédure permettant de résoudre le système d'équations suivant :

    a.x + b.y = c

    d.x + e.y = f

SERIE 3 – Les Tableaux

12. Soit V un vecteur monodimensionnel de 5 entiers, écrire les procédures suivantes :
    a. Procédure sans paramètre permettant de calculer la somme des 5 entiers,
    b. Procédure sans paramètre permettant de calculer le produit des 5 entiers,
    c. Procédure sans paramètre permettant de calculer la moyenne des 5 entiers,
    d. Procédure sans paramètre permettant de trouver la valeur max entre les 5 entiers,
    e. Procédure sans paramètre permettant de trouver la valeur min entre les 5 entiers,
    f. Procédure sans paramètre permettant de calculer la somme, le produit, la moyenne, le max et le min des 5 entiers.
    g. Procédure sans paramètre permettant de trier les 5 entiers dans l'ordre croissant
    h. Procédure sans paramètre permettant de trier les 5 entiers dans l'ordre décroissant.

13. Ecrire une fonction prenant en paramètre un vecteur V, sa taille et une position p. Cet algorithme doit vous permettre de retourner l'indice de la valeur min du vecteur en partant de la position p.

14. Ecrire une procédure sans paramètre, permettant de trier un vecteur d'entiers en s'appuyant sur la fonction de la question 14.

15. Soient U et V deux vecteurs d'entiers, lu et lv leurs tailles respectives.
    a. On suppose que U et V sont déjà triés, écrire une procédure sans paramètre, permettant de fusionner ces deux vecteurs pour obtenir un troisième vecteur trié.
    b. Ecrire une fonction permettant de fusionner les deux vecteurs en un seul.
       Prototype : int * FusionVecteurs(int *U, int lu, int *V, int lv)
    c. On suppose que les deux vecteurs U et V sont déjà triés, écrire une fonction permettant de fusionner les deux vecteurs en un seul en effectuant un remplissage du nouveau vecteur dans un ordre croissant. (Remarque : à la fin du remplissage du nouveau vecteur, celui-ci va se retrouver trié également)
       Prototype : int * FusionCroissanteVecteurs(int *U, int lu, int *V, int lv)

SERIE 4 – Les Tableaux (suite)

16. Ecrire une procédure permettant de demander à l'utilisateur de fournir la taille d'un vecteur d'entiers, puis de saisir les valeurs pour ce vecteur avant d'afficher la liste de ces valeurs.

17. Ecrire une fonction permettant en paramètre la taille d'un vecteur d'entiers et retournant l'adresse mémoire de ce vecteur.

18. Ecrire une procédure prenant en paramètre l'adresse d'un vecteur d'entiers et sa taille, et permettant de saisir les éléments pour ce vecteur.

19. Ecrire une procédure prenant en paramètre l'adresse d'un vecteur d'entiers et sa taille, et permettant d'afficher à l'écran les éléments de ce vecteur.

20. Ecrire une procédure permettant de calculer la somme de deux matrices de taille 2x3.

21. Ecrire une procédure permettant de calculer le produit de deux matrices de tailles 4x3 pour T et 3x2 pour V.

SERIE 5 – Les Chaînes de Caractères

22. Trouver dans un tableau de caractères toutes les sous-chaînes de caractères de longueur maximale dont la première lettre et la seconde sont identiques. Affichez le nombre de sous-chaînes trouvées.

23. Trouver dans un tableau de caractères toutes les sous-chaînes de caractères dont la première lettre et la seconde sont identiques. Affichez le nombre de sous-chaînes trouvées.

24. Ecrire la fonction « **PositionChaine »** permettant de rechercher la position de la première occurrence d'une chaine « **t** » à l'intérieure d'une autre chaine « **s** ». Si la chaine « **t** » n'existe pas à l'intérieur de la chaine « **s** », la fonction renvoie la valeur **-1**.
    Prototype : **int** PositionChaine (**char** ***** s , **char *** t)

25. Ecrire la fonction « **PositionChaineIndice »** permettant de rechercher la première position d'une chaine « **t** » à l'intérieure d'une autre chaine « **s** » à partir d'un indice donné. Si la chaine « **t** » n'existe pas à l'intérieur de la chaine « **s** » à partir de cet indice, la fonction renvoie la valeur -**1**.
    Prototype : **int** PositionChaineIndice (**char** ***** s , **char *** t , **unsigned int** pos)

26. Ecrire une procédure « **PositionsChaine »** permettant de rechercher et d'afficher toutes les positions d'une chaine « **t** » à l'intérieure d'une autre chaine « **s** ».
    Prototype : **void** PositionsChaine (**char** ***** s , **char *** t)

27. Ecrire la fonction « ListePositionsChaine » permettant retourner un tableau contenant toutes les positions des occurrences de la chaine « t » à l'intérieure de la chaine « s ».
    Prototype : **int *** ListePositionsChaine (**char *** s , **char *** t , **int *** nb_occur)

28. Adapter la fonction « ListePositionsChaine » (exo 28) pour retourner un tableau contenant toutes les positions du mot « t » ou de l'occurrence « t » à l'intérieure de la chaine « s ».
    Prototype : **int *** ListePositionsChaine (**char *** s , **char *** t , **int *** nb_occur , **char** opt)

29. Ecrire la fonction « SaisirChaine » permettant de saisir une chaine de caractères au clavier et lui allouant la taille mémoire exacte sans pour autant connaitre sa taille à l'avance.
    Prototype : **char *** SaisirChaine ()

SOLUTIONS

30. Soient les procédures suivantes :

Algorithme Procedure P1( )	Algorithme Procedure P2( )
Var A, B, C **: Entier ;** **Debut** **Lire (**A , B**)** ; C ← A ; A ← B ; B ← C ; **Afficher (**A , B**)** ; **Fin**	**Var** A, B **: Entier ;** **Debut** **Lire (**A , B**)** ; A ← A + B ; B ← A - B ; A ← A - B ; **Afficher (**A , B**)** ; **Fin**

d) Remplissez les cellules grisées sur les tableaux de valeurs ci-dessous.

e) Quel rôle joue chacune des procédures ?

f) Laquelle de ces procédures est la plus rapide ?

Procédure P1				
Entrée			Sortie	
A	B		A	B
0	2	→	2	0
-1	1		1	-1
4	2		2	4

Procédure P2				
Entrée			Sortie	
A	B		A	B
0	2	→	2	0
-1	1		1	-1
4	2		2	4

31. Ecrire une procédure permettant d'afficher la valeur max entre deux entiers qu'un utilisateur saisit au clavier.

```c
void exo_2_v1()
{
    int a,b;

    printf("Saisir deux entiers : ");
    scanf("%d %d",&a,&b);
    getchar();

    if(a > b)
    {
        printf("la valeur maximal est : %d",a);
    }
    else
    {
        printf("la valeur maximal est : %d",b);
    }
}




void exo_2_v2()
```

```c
{
    int a,b;

    printf("Saisir deux entiers : ");
    scanf("%d %d",&a,&b);
    getchar();

    printf("la valeur maximal est : %d",(a>b)? a:b);
}
```

32. Ecrire une fonction permettant de retourner la valeur max entre deux entiers qu'un utilisateur saisit au clavier.

```c
int exo_3_v1()
{
    int a,b;

    printf("Saisir deux entiers : ");
    scanf("%d %d",&a,&b);
    getchar();

    if(a > b)
    {
        printf("la valeur maximal est : %d",a);
        return a ;
    }
    else
    {
        printf("la valeur maximal est : %d",b);
        return b ;
    }
}

int exo_3_v2()
{
    int a,b;

    printf("Saisir deux entiers : ");
    scanf("%d %d",&a,&b);
    getchar();

    return (a > b)? a:b ;
}
```

33. Ecrire une procédure prenant en paramètre deux variables de type entier et permettant d'afficher la valeur max entre ces deux variables.

```c
void exo_4(int a, int b)
{
    printf("\n le max est : %d\n",(a < b)? b:a);
}
```

34. Ecrire une fonction prenant en paramètre deux variables de type entier et permettant de retourner la valeur max entre ces deux variables.

```
int exo_5(int a, int b)
{
    return (a < b)? b:a ;
}
```

35. Refaire les exercices 2, 3, 4 et 5 mais cette fois-ci en prenant en compte trois valeurs entières.

```
void exo_6_2_v1()
{
  int a,  b, c ;
  printf ("saisissez  trois entiers : ");
  scanf("%d %d %d", &a, &b, &c);
  getchar();
  printf("\nle max est %d \n",((((a > b)? a:b) > c )? ((a > b)? a:b):c)) ;
}

void exo_6_2_v2()
{
  int a,  b, c,d ;
  printf ("saisissez  trois entiers : ");
  scanf("%d %d %d", &a, &b, &c);
  getchar();

  printf("\nle max est %d \n",((d=((a > b)? a:b) > c )? d:c)) ;
}

void exo_6_2_v3()
{
    int a,  b, c ;

    printf ("saisissez  trois entiers : ");
    scanf("%d %d %d",&a, &b, &c);
    getchar();

    if((a > b)  &&  (a > c))
      printf("\nla valeur maximale est : %d\n",a);
    else if ((b > a)  &&  ( b > c))
        printf("\nla valeur maximale est : %d\n",b);
    else
        printf("\nla valeur maximale est : %d\n",c);
}

void exo_6_2_v4()
{
    int a,  b, c ;

    printf ("saisissez  trois entiers : ");
    scanf("%d %d %d",&a, &b, &c);
    getchar();

    printf("\nle max est %d \n",exo_5(c, exo_5(a, b))) ;
}
```

```c
void exo_6_3_v1(int a, int b, int c)
{
    if((a > b)  &&  (a > c))
      printf("\nla valeur maximale est : %d\n",a);
    else if ((b > a)  &&  ( b > c))
        printf("\nla valeur maximale est : %d\n",b);
    else
        printf("\nla valeur maximale est : %d\n",c);
}


int exo_6_4_v1(int a,int  b,int c)
{
    return exo_5(c, exo_5(a, b)) ;
}


int exo_6_4_v2(int a, int b,int c)
{
  return (((a > b)? a:b) > c )? ((a > b)? a:b):c ;
}


int exo_6_4_v3(int a,int  b,int c)
{
    if((a > b)  &&  (a > c))
      return a;
    else if ((b > a)  &&  ( b > c))
       return b;
    else
        return c;
}
```

36. Soit la fonction f(n) = n! avec (0! = 1 et n! = n*(n-1)! Pour tout n > 0)

h) Ecrire une procédure sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et d'afficher la valeur de f(n).

```c
void exo_7_a_v1()
{
    int n, x = 1, y;

    printf("\n saisir une valeur entière : ");
    scanf("%d" , &n);
    getchar();

    if(n < 0)
    {
        printf("\n l'entier doit être positif\n");
        return;
    }
    if(n > 12)
    {
        printf("\n le résultat dépasse la capacite d'un entier sur 4 octets\n");
        return;
    }
    y = n;

    if((n == 0) || (n == 1)){
        printf("\n le factoriel de %d est %d \n", n, 1);
        return;
    }

    do
    {
        x = x*y;
        y = y-1;
    }
    while(y !=1 );

    printf("\n le factoriel de %d vaut : %d\n", n, x);
}

void exo_7_a_v2()
{
    int n,fact = 1, i ;

    printf("\n saisir une valeur entière : ");
    scanf("%d",&n);
    getchar();

    if(n < 0)
    {
        printf("\n l'entier doit être positif\n") ;
        return  ;
    }
```

```
    if(n > 12)
    {
        printf("\n le résultat dépasse la capacite d'un entier sur 4 octets\n");
        return;
    }

    for(i = 1 ; i <= n ; i++)
    {
        fact = fact * i ;
    }
    printf("\n le factoriel de %d vaut : %d\n",n,fact);
}
```

i) Ecrire une procédure avec comme paramètre la variable n, permettant d'afficher la valeur de f(n).

```
void exo_7_b(int n)
{
    int fact = 1, i ;

    if(n < 0)
    {
        printf("\n l'entier doit être positif\n") ;
        return ;
    }
    if(n > 12)
    {
        printf("\n le résultat dépasse la capacite d'un entier sur 4 octets\n");
        return;
    }
    for(i = 1 ; i <= n ; i++)
    {
        fact = fact * i ;
    }
    printf("\n le factoriel de %d vaut : %d\n", n, fact);
}
```

j) Ecrire une fonction sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et de retourner la valeur de f(n).

```
int exo_7_c_v1()
{
    int n = 13, i, fact = 1;

    while((n > 12) || (n  < 0))
    {
        printf("saisir une valeur entière entre 0 et 12 : ");
        scanf("%d" , &n);
        getchar();
    }

    for(i = 1 ; i <= n ; i++)
    {
        fact = fact * i ;
    }
    printf("\n le factoriel de %d vaut : %d\n", n, fact);
    return fact;
}
```

```c
int exo_7_c_v2(int * flag)
{
    int n, i, fact = 1;
    *flag = 1;

    printf(" saisir une valeur entière entre 0 et 12 : ");
    scanf("%d" , &n);
    getchar();

    if((n > 12) || (n < 0))
    {
        *flag = 0;
        return 0;
    }

    for(i = 1 ; i <= n ; i++)
    {
        fact = fact * i ;
    }

    return fact;
}
```

k) Ecrire une fonction avec comme paramètre la variable n, permettant de retourner la valeur de f(n).

```c
int exo_7_d(int n , int *flag)
{
    int i, fact = 1;
    *flag = 1;

    if((n <0) || n>12 )
    {
        *flag = 0;
        return 0;
    }

    for(i = 1 ; i <= n ; i++)
    {
        fact = fact * i ;
    }

    return fact;
}
```

l) Ecrire une procédure avec comme paramètres les variables n et fact, permettant de calculer la valeur de f(n) et stocker le résultat dans l'adresse de fact.

```c
void exo_7_e(int n , int * fact ,  int * flag)
{
    int i ;
    *fact = 1;
    *flag = 1;

    if((n < 0) || (n > 12 ))
    {
        *flag = 0;
        return ;
```

```
    }

    for(i = 1 ; i <= n ; i++)
    {
        *fact = *fact * i ;
    }
}
```

m) Ecrire une procédure, permettant de trouver la plus petite valeur de n ainsi que celle de f(n) pour laquelle f(n) >= 121.

```
void exo_7_f()
{
    int n = 0, i, fact = 1;

    while(fact < 121)
    {
       fact *= ++n;
    }
    printf("\n le fact de %d est %d\n", n, fact);
}
```

n) Ecrire une fonction récursive avec comme paramètre la variable n et permettant de retourner la valeur de f(n).

```
int exo_7_g(int n , int * flag)
{
    if((n > 12) || (n < 0))
    {
        *flag = 0;
        return 0;
    }

    if((n == 0) || (n == 1))
    {
       *flag = 1;

        return 1;
    }

    return n * exo_7_g(n-1 , flag);
}
```

37. Soit la suite U(n) = 3*U(n - 1) + 2 pour tout n > 0 avec U(0) = 1

h. Ecrire une procédure sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et d'afficher la valeur de U(n).

```c
void exo_8_a()
{
    int n, u = 1, i;

    printf("\n saisir une valeur entière : ");
    scanf("%d" , &n);
    getchar();

    if(n < 0)
    {
        printf("\n l'entier doit être positif\n");
        return;
    }

    if (n > 18)
    {
        printf("\n le résultat  de la suite  dépasse la limites de type entier    \n");
        return ;
    }

    for(i = 0 ; i < n ; i++)
    {
        u = 3*u + 2 ;
    }
    printf("\n u(%d) = %d \n", n , u);
}
```

i. Ecrire une procédure avec comme paramètre la variable n, permettant d'afficher la valeur de U(n).

```c
void exo_8_b(int n)
{
    int u = 1, i;

    if (n > 18)
    {
        printf("\n le résultat  de la suite  dépasse la limites de type entier\n");
        return ;
    }
    if (n < 0)
    {
        printf("\n l'entier doit être positif\n");
        return;
    }
    for(i = 0 ; i < n ; i++)
    {
        u = 3*u + 2 ;
    }
    printf("\n u(%d) = %d \n", n , u);
}
```

j. Ecrire une fonction sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et de retourner la valeur de U(n).

```c
int exo_8_c_v1()
{
    int n = -1, u = 1, i;

    while ((n < 0) || (n > 18))
    {
        printf(" saisir une valeur entière entre 0 et 18 : ");
        scanf("%d" , &n);
        getchar();
    }
    for(i = 0 ; i < n ; i++)
    {
        u = 3*u + 2 ;
    }
    return u;
}

int exo_8_c_v2(int * flag)
{
    int n, u = 1, i;
    *flag = 1 ;

    printf(" saisir une valeur entière entre 0 et 18 : ");
    scanf("%d" , &n);
    getchar();

    if ((n < 0) || (n > 18))
    {
        *flag = 0 ;
        return 0;
    }
    for(i = 0 ; i < n ; i++)
    {
        u = 3*u + 2 ;
    }
    return u;
}
```

k. Ecrire une fonction avec comme paramètre la variable n, permettant de retourner la valeur de U(n).

```c
int exo_8_d(int n , int *flag)
{
    int u = 1, i;
    *flag = 1 ;

    if ((n < 0) || (n > 18))
    {
        *flag = 0 ;
        return 0;
    }
    for(i = 0 ; i < n ; i++)
    {
        u = 3*u + 2 ;
    }
    return u;
}
```

l. Ecrire une procédure, permettant de trouver la plus petite valeur de n ainsi que celle de U(n) pour laquelle U(n) >= 121.

```
void exo_8_e()
{
    int n = 0, u = 1 ;

    while(u < 121)
    {
       u = 3*u + 2 ;
       n++ ;
    }
    printf("\n u(%d) : %d\n", n , u);
}
```

m. Ecrire une fonction récursive avec comme paramètre la variable n et permettant de retourner la valeur de U(n).

```
int exo_8_f(int n , int * flag)
{
    if((n < 0) || (n > 18))
    {
        *flag = 0;
        return 0;
    }
    if(n == 0)
    {
       *flag = 1;
        return 1;
    }
    return 3 * exo_8_f(n-1 , flag) + 2 ;
}
```

n. Ecrire une procédure récursive avec comme paramètres la variable n et une autre variable passée par adresse et permettant de retourner la valeur de U(n).

```
void exo_8_g(int n , int * res , int * flag)
{
    if((n < 0) || (n > 18))
    {
    *flag = 0;
    *res = 0;
        return ;
    }
    if(n == 0)
    {
        *flag = 1;
        *res = 1;
        return ;
    }
    exo_8_g(n - 1 , res , flag)  ;
    *res = 3 * (*res) + 2 ;
}
```

38. Soit la série suivante :

$$S(n) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \quad avec\ n\ entier > 0$$

h. Ecrire une procédure sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et d'afficher la valeur de S(n).

```c
void exo_9_a()
{
    int n,i;
    float s ;

    printf("\n saisir une valeur entière : ");
    scanf("%d" , &n);
    getchar();

    if (n <= 0)
    {
        printf("\n le résultat  de la suite  n'est pas définie\n");
        return ;
    }

    for(s = 0.0 , i = 1 ; i <= n ; i++)
    {
        s += 1.0/i ;
    }
    printf("\n S(%d) = %.2f \n", n, s);
}
```

i. Ecrire une procédure avec comme paramètre la variable n, permettant d'afficher la valeur de S(n).

```c
void exo_9_b(int n)
{
    int i ;
    float s ;

    if (n <= 0)
    {
        printf("\n Le résultat  de la suite  n'est pas définie\n");
        return ;
    }

    for(s = 0.0 , i = 1 ; i <= n ; i++)
    {
        s += 1.0/i ;
    }
    printf("\n S(%d) = %.2f \n", n, s);
}
```

j. Ecrire une fonction sans paramètre, permettant de demander à l'utilisateur de saisir la valeur de n et de retourner la valeur de S(n).

```c
float exo_9_c(int * flag)
{
    int n , i;
    float s ;
    *flag = 0 ;

    printf("\n saisir une valeur entière : ");
    scanf("%d" , &n);
    getchar();

    if (n <= 0)
    {
        return 0 ;
    }
    for(s = 0.0 , i = 1 ; i <= n ; i++)
    {
        s += 1.0/i ;
    }
    *flag = 1 ;
    return s ;
}
```

k. Ecrire une fonction avec comme paramètre la variable n, permettant de retourner la valeur de S(n).

```c
float exo_9_d(int n , int * flag)
{
    int i;
    float s ;
    *flag = 0 ;

    if (n <= 0)
    {
        return 0 ;
    }
    for(s = 0.0 , i = 1 ; i <= n ; i++)
    {
        s += 1.0/i ;
    }
    *flag = 1 ;
    return s ;
}
```

l. Ecrire une procédure, permettant de trouver la plus petite valeur de n ainsi que celle de S(n) pour laquelle S(n) >= 3.

```c
void exo_9_e()
{
    int n = 0 ;
    float s = 0.0 ;

    while(s < 3)
    {
       s += 1.0/++n ;
    }
    printf("\n S(%d) : %f\n", n, s);
}
```

m. Ecrire une fonction récursive avec comme paramètre la variable n et permettant de retourner la valeur de S(n).

```
float exo_9_f(int n , int *flag)
{
    if(n <= 0)
    {
        *flag = 0;
        return 0;
    }
    if(n == 1)
    {
        *flag = 1;
        return 1;
    }

    return 1.0/n + exo_9_f(n-1 ,f lag) ;
}
```

n. Ecrire une procédure récursive avec comme paramètres la variable n et une autre variable passée par adresse et permettant de retourner la valeur de S(n).

```
void exo_9_g(int n , float * res , int * flag)
{
    if(n <= 0)
    {
        *flag = 0;
        *res = 0;
        return ;
    }
    if(n == 1)
    {
        *flag = 1;
        *res = 1;
        return ;
    }
    exo_9_g(n - 1 , res , flag) ;
    *res = 1.0/n + (*res) ;
}
```

39. Ecrire une procédure permettant de résoudre l'équation de second degré $f(x) = a.x^2 + b.x + c = 0$

```c
#include<stdio.h>
#include<math.h>

void exo_10()
{
    float a, b, c, delta ;

    printf("\n saisir les valeurs de a, b et c : ")  ;
    scanf("%f%f%f", &a, &b, &c) ;
    getchar() ;

    if(a == 0)
    {
        if(b == 0)
        {
            if(c == 0)
            {
                printf("\n il existe une infinité de solutions\n") ;
            }
            else
            {
                printf("\n pas de solution\n") ;
            }
        }
        else
        {
            printf("\n une solution : x = %.2f\n", -c/b) ;
        }
    }
    else
    {
        delta = pow(b,2) - 4*a*c ;
        if(delta < 0)
        {
           printf("\n pas de solution\n") ;
        }
        else if(delta == 0)
        {
            printf("\n une solution : x = %.2f\n", -b/(2*a)) ;
        }
        else
        {
            printf("\n deux solutions : x1 = %.2f et x2 = %.2f\n",
                                (-b - sqrt(delta))/(2*a), (-b + sqrt(delta))/(2*a)) ;
        }
    }
}
```

40. Ecrire une procédure permettant de résoudre le système d'équations suivant :

a.x + b.y = c

d.x + e.y = f

```
void exo_11()
{
    float a, b, c, d, e, f, det, detx, dety ;

    printf("\n saisir les valeurs de a, b, c, d, e et f : ")  ;
    scanf("%f%f%f%f%f%f", &a, &b, &c, &d, &e, &f) ;
    getchar() ;

    det  = a*e - b*d ;
    detx = c*e - b*f ;
    dety = a*f - c*d ;

    if(det == 0)
    {
        if((a == 0) || (b == 0) || (c == 0) || (d == 0) || (e == 0) || (f == 0))
        {
            printf("\n pas de solution\n") ;
        }
        else if((a/d == b/e) && (a/d ==  c/f))
        {
            printf("\n une solution : y = %.2fx + %.2f\n",-a/b, c/b) ;
        }
        else
        {
            printf("\n pas de solution\n") ;
        }
    }
    else
    {
        printf("\n deux solutions : x = %.2f et y = %.2f\n", detx/det, dety/det) ;
    }
}
```

Solutions Série 3 – Les Tableaux

```c
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<math.h>
#define X 5
#define Y 7
```

41. Soit V un vecteur monodimensionnel de 5 entiers, écrire les procédures suivantes :

i. Procédure sans paramètre permettant de calculer la somme des 5 entiers,
j. Procédure sans paramètre permettant de calculer le produit des 5 entiers,
k. Procédure sans paramètre permettant de calculer la moyenne des 5 entiers,
l. Procédure sans paramètre permettant de trouver la valeur max entre les 5 entiers,
m. Procédure sans paramètre permettant de trouver la valeur min entre les 5 entiers,
n. Procédure sans paramètre permettant de calculer la somme, le produit, la moyenne, le max et le min des 5 entiers.
o. Procédure sans paramètre permettant de trier les 5 entiers dans l'ordre croissant
p. Procédure sans paramètre permettant de trier les 5 entiers dans l'ordre décroissant.

```c
void exo_13_a()
{
    int i, som = 0 , T[X] = {15 , 3 , -5 , 12 , 36} ;

    for(i = 0 ; i < X ; i++)
    {
        som += T[i];
    }
    printf("\n La Somme du vecteur est : %d\n", som);
}


void exo_13_b()
{
    int i, prod = 1 , T[X] = {15 , 3 , -5 , 12 , 36} ;

    for(i = 0 ; i < X ; i++)
    {
        prod *= T[i];
    }
    printf("\n La produit du vecteur est : %d\n", prod);
}
```

```c
void exo_13_c()
{
    int i, som = 0 , T[X] = {15 , 3 , -5 , 12 , 36} ;

    for(i = 0 ; i < X ; i++)
    {
        som += T[i];
    }
    printf("\n La moyenne du vecteur est : %f\n", som/5.0);
}


void exo_13_d()
{
    int i, T[X] = {15 , 3 , -5 , 12 , 36}, max = T[0] ;

    for(i = 1 ; i < X ; i++)
    {
        if(T[i] > max)
            max = T[i] ;
    }
    printf("\n Le max du vecteur est : %d\n", max);
}

void exo_13_e()
{
    int i, T[X] = {15 , 3 , -5 , 12 , 36}, min = T[0] ;

    for(i = 1 ; i < X ; i++)
    {
        if(T[i] < min)
            min = T[i] ;
    }
    printf("\n Le min du vecteur est : %d\n", min);
}

void exo_13_f()
{
    int i, T[X] = {15 , 3 , -5 , 12 , 36}, min = T[0], max = T[0], som = T[0], prod =
T[0] ;

    for(i = 1 ; i < X ; i++)
    {
        if(T[i] < min)
            min = T[i] ;
        if(T[i] > max)
            max = T[i] ;
        som += T[i] ;
        prod *= T[i] ;
    }
    printf("\n Le min du vecteur est : %d\n", min);
    printf("\n Le max du vecteur est : %d\n", max);
    printf("\n La somme du vecteur est : %d\n", som);
    printf("\n Le produit du vecteur est : %d\n", prod);
    printf("\n La moyenne du vecteur est : %f\n", som/5.0);
}
```

```c
void exo_13_f_bis()
{
    int i, min, max, som, prod , T[Y]  ;

    for(i = 0 ; i < Y ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", T+i ); // scanf("%d", &T[i]) ;
        getchar();
    }

    min = T[0];
    max = T[0];
    som = T[0];
    prod = T[0] ;

    for(i = 1 ; i < Y ; i++)
    {
        if(T[i] < min)
            min = T[i] ;
        if(T[i] > max)
            max = T[i] ;
        som += T[i] ;
        prod *= T[i] ;
    }
    printf("\n Le min du vecteur est : %d\n", min);
    printf("\n Le max du vecteur est : %d\n", max);
    printf("\n La somme du vecteur est : %d\n", som);
    printf("\n Le produit du vecteur est : %d\n", prod);
    printf("\n La moyenne du vecteur est : %f\n", som/5.0);
}

void exo_13_g()
{
    int i, j, aux, T[Y] ;

    for(i = 0 ; i < Y ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", &T[i]) ;
        getchar();
    }

    for(i = 0 ; i < Y -1 ; i++)
    {
        for(j = i + 1 ; j < Y ; j++)
        {
            if(T[i] > T[j])
            {
                aux = T[i] ;
                T[i] = T[j] ;
                T[j] = aux ;
            }
        }
    }
    printf("\n_____\n") ;
    for(i = 0 ; i < Y ; i++)
    {
        printf("\n T[%d] : %d\n", i, T[i]);
    }
}
```

```c
void exo_13_h()
{
    int i, j, aux, T[Y] ;

    for(i = 0 ; i < Y ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", &T[i]) ;
        getchar();
    }

    for(i = 0 ; i < Y -1 ; i++)
    {
        for(j = i + 1 ; j < Y ; j++)
        {
            if(T[i] < T[j])
            {
                aux = T[i] ;
                T[i] = T[j] ;
                T[j] = aux ;
            }
        }
    }
    printf("\n_____\n") ;
    for(i = 0 ; i < Y ; i++)
    {
        printf("\n T[%d] : %d\n", i, T[i]);
    }
}
```

42. Ecrire une fonction prenant en paramètre un vecteur V, sa taille et une position p. Cet algorithme doit vous permettre de retourner l'indice de la valeur min du vecteur en partant de la position p.

```c
int exo_14(int *V , int N , int p)
{
    int i = p, ind_min = p ;

    if((N <= 0) || (p >= N))
        return -1 ;

    for(i = p + 1 ; i < N ; i++)
    {
        if(V[i] < V[ind_min])
        {
            ind_min = i;
        }
    }
    return ind_min;
}

void exo_14_test()
{
    int i, pos, T[Y] ;

    for(i = 0 ; i < Y ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
```

```
        scanf("%d", &T[i]) ;
        getchar();
    }
    pos = exo_14(T , Y , 6) ;
    if(pos < 0)
        printf("\n Erreur de taille ou d'indice\n") ;
    else
        printf("\n Position : %d\n", pos) ;
}
```

43. Ecrire une procédure sans paramètre, permettant de trier un vecteur d'entiers en s'appuyant sur la fonction de la question 14.

```
void exo15 ()
{
    int i, T[Y], aux, pos ;

    for(i = 0 ; i < Y ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", &T[i]) ;
        getchar();
    }

    for(i = 0 ; i < Y ; i++)
    {
        pos = exo_14(T , Y , i);
        aux = T[i];
        T[i] = T[pos]  ;
        T[pos] = aux;
    }

    printf("\n_____\n") ;
    for(i = 0 ; i < Y ; i++)
    {
        printf("\n T[%d] : %d\n", i, T[i]);
    }
}
```

44. Soient U et V deux vecteurs d'entiers, lu et lv leurs tailles respectives.

    d. On suppose que U et V sont déjà triés, écrire une procédure sans paramètre, permettant de fusionner ces deux vecteurs pour obtenir un troisième vecteur trié.

    e. Ecrire une fonction permettant de fusionner les deux vecteurs en un seul.

    Prototype : int * FusionVecteurs(int *U, int lu, int *V, int lv)

    f. On suppose que les deux vecteurs U et V sont déjà triés, écrire une fonction permettant de fusionner les deux vecteurs en un seul en effectuant un remplissage du nouveau vecteur dans un ordre croissant. (Remarque : à la fin du remplissage du nouveau vecteur, celui-ci va se retrouver trié également)

    Prototype : int * FusionCroissanteVecteurs(int *U, int lu, int *V, int lv)

```c
void exo16 ()
{
    int i = 0 , j = 0 , k = 0 ;
    int U[5] = {4, 12, 18, 20 , 32} ;
    int V[7] = {1, 5 , 8 , 20 , 22 , 25 , 30} ;
    int W[12];

    while((i < 5) && (j < 7))
    {
        if(U[i] < V[j])
        {
            W[k++] = U[i++];
        }
        else
        {
            W[k++] = V[j++];
        }
    }

    while(i < 5)
    {
        W[k++] = U[i++];
    }
    while(j < 7)
    {
        W[k++] = V[j++];
    }
    printf("\n_____\n") ;
    for(i = 0 ; i < 12 ; i++)
    {
        printf("\n  W[%d] : %d\n", i, W[i]);
    }
}

int * FusionVecteurs(int *U, int lu, int *V, int lv)
{
    int i;
    int *F = malloc((lu+lv)*sizeof(int));
    for(i = 0 ; i < lu ; i++)
    {
        F[i] = U[i];
    }
    for(i = 0 ; i < lv ; i++ )
    {
        F[lu + i] = V[i];
    }
    return F;
}

void FusionVecteurs_test ()
{
    int i ;
    int U[5] = {4, 12, 18, 20 , 32} ;
    int V[7] = {1, 5 , 8 , 20 , 22 , 25 , 30} ;
    int *W = FusionVecteurs(U, 5, V, 7);
    printf("\n_____\n") ;
    for(i = 0 ; i < 12 ; i++)
    {
        printf("\n  W[%d] : %d\n", i, W[i]);
    }
}
```

```c
int * FusionCroissanteVecteurs(int *U, int lu, int *V, int lv)
{
    int i = 0 , j = 0, k = 0 ;
    int *W = malloc ((lu+lv)*sizeof(int));

    while((i < lu) && (j < lv))
    {
        if(U[i] < V[j])
        {
            W[k++] = U[i++];
        }
        else
        {
            W[k++] = V[j++];
        }
    }

    while(i < lu)
    {
        W[k++] = U[i++];
    }
    while(j < lv)
    {
        W[k++] = V[j++];
    }
    return W;
}

void FusionCroissanteVecteurs_test ()
{
    int i ;
    int U[5] = {4, 12, 18, 20 , 32} ;
    int V[7] = {1, 5 , 8 , 20 , 22 , 25 , 30} ;
    int *W = FusionCroissanteVecteurs(U, 5, V, 7);
    printf("\n_____\n") ;
    for(i = 0 ; i < 12 ; i++)
    {
        printf("\n  W[%d] : %d\n", i, W[i]);
    }
}
```

45. Ecrire une procédure permettant de demander à l'utilisateur de fournir la taille d'un vecteur d'entiers, puis de saisir les valeurs pour ce vecteur avant d'afficher la liste de ces valeurs.

```c
void exo_17_1()
{
    int i, taille = 0, *T ;

    while(taille <= 0)
    {
        printf("\n Saisir la taille du vecteur : ") ;
        scanf("%d", &taille);
        getchar();
    }

    T = malloc(taille*sizeof(int)) ;

    for(i = 0 ; i < taille ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", &T[i]) ;
        getchar();
    }
    printf("\n_____\n") ;
    for(i = 0 ; i < taille ; i++)
    {
        printf("\n  T[%d] : %d\n", i, T[i]);
    }
}


int * exo_17_2(int * taille)
{
    int i, *T ;
    *taille = 0 ;
    while(*taille <= 0)
    {
        printf("\n Saisir la taille du vecteur : ") ;
        scanf("%d", &taille);
        getchar();
    }

    T = malloc(*taille*sizeof(int)) ;

    for(i = 0 ; i < *taille ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", &T[i]) ;
        getchar();
    }
    printf("\n_____\n") ;
    for(i = 0 ; i < *taille ; i++)
    {
        printf("\n  T[%d] : %d\n", i, T[i]);
    }
}
```

46. Ecrire une fonction permettant en paramètre la taille d'un vecteur d'entiers et retournant l'adresse mémoire de ce vecteur.

```
int * allouer_vecteur_dynamique(int taille)
{
    if(taille <= 0)
        return NULL ;

    return malloc(taille*sizeof(int));
}
```

47. Ecrire une procédure prenant en paramètre l'adresse d'un vecteur d'entiers et sa taille, et permettant de saisir les éléments pour ce vecteur.

```
void saisir_vecteur(int * T, int taille)
{
    int i;
    for(i = 0 ; i < taille ; i++)
    {
        printf("\n Saisir T[%d] : ", i);
        scanf("%d", &T[i]) ;
        getchar();
    }
}
```

48. Ecrire une procédure prenant en paramètre l'adresse d'un vecteur d'entiers et sa taille, et permettant d'afficher à l'écran les éléments de ce vecteur.

```
void afficher_vecteur(int * T, int taille)
{
    int i ;
    printf("\n_____\n") ;
    for(i = 0 ; i < taille ; i++)
    {
        printf("\n T[%d] : %d\n", i, T[i]);
    }
    printf("\n_____\n") ;
}

void Test_vecteurs()
{
    int i, taille = -100, *T ;

    while(taille <= 0)
    {
        printf("\n Donnez la taille du vecteur : ") ;
        scanf("%d", &taille);
        getchar() ;
    }
    T = allouer_vecteur_dynamique(taille) ;
    saisir_vecteur(T, taille) ;
    afficher_vecteur(T, taille) ;
}
```

49. Ecrire une procédure permettant de calculer la somme de deux matrices de taille 2x3.

```c
void somme_matrices_statiques()
{
    int T[2][3], V[2][3], S[2][3] ;
    int i, j;

    printf("\n saisir la matrice T : \n");

    for(i = 0 ; i < 2 ; i++)
    {
        for(j = 0 ; j < 3 ; j++)
        {
            printf("donner T[%d][%d] : ", i, j);
            scanf("%d", &T[i][j]);
            getchar();
        }
    }

    printf("\n saisir la matrice V : \n");

    for(i = 0 ; i < 2 ; i++)
    {
        for(j = 0 ; j < 3 ; j++)
        {
            printf("donner V[%d][%d] : ",i, j);
            scanf("%d",&V[i][j]);
            getchar();
        }
    }

    for(i = 0 ; i < 2 ; i++)
    {
        for(j = 0 ; j < 3 ; j++)
        {
            S[i][j] = T[i][j] + V[i][j];
        }
    }

    printf("\n Affichage de la somme des matrices : \n");

    for(i = 0 ; i < 2 ; i++)
    {
        for(j = 0 ; j < 3 ; j++)
        {
            printf(" %d ", S[i][j]);
        }
        printf("\n");
    }
}
```

50. Ecrire une procédure permettant de calculer le produit de deux matrices de tailles 4x3 pour T et 3x2 pour V.

```c
void produit_matrices_statiques()
{
    int T[4][3], V[3][2], P[4][2] ;
    int i, j, k;

    printf("\n saisir la matrice T : \n");

    for(i = 0 ; i < 4 ; i++)
    {
        for(j = 0 ; j < 3 ; j++)
        {
            printf("donner T[%d][%d] : ", i, j);
            scanf("%d", &T[i][j]);
            getchar();
        }
    }

    printf("\n saisir la matrice V : \n");

    for(i = 0 ; i < 3 ; i++)
    {
        for(j = 0 ; j < 2 ; j++)
        {
            printf("donner V[%d][%d] : ",i, j);
            scanf("%d",&V[i][j]);
            getchar();
        }
    }

    for(i = 0 ; i < 4 ; i++)
    {
        for(j = 0 ; j < 2 ; j++)
        {
            P[i][j] = 0 ;
            for(k = 0 ; k < 3 ; k++)
            {
                P[i][j] += T[i][k] * V[k][j] ;
            }
        }
    }

    printf("\n Affichage du produit des matrices : \n");

    for(i = 0 ; i < 4 ; i++)
    {
        for(j = 0 ; j < 2 ; j++)
        {
            printf(" %d ", P[i][j]);
        }
        printf("\n");
    }
}
```

Solutions Série 5 – Les Chaînes de Caractères

51. Trouver dans un tableau de caractères toutes les sous-chaînes de caractères de longueur maximale dont la première lettre et la seconde sont identiques. Affichez le nombre de sous-chaînes trouvées.

```c
void exo_23_v1(char *str)
{
    int i = 0, nbr_occur = 0;

    if(str[i] != '\0')
    {
        while(str[i+1] != '\0')
        {
            if (str[i] == str[i+1])
            {
                printf("\n %s",str+i);
                nbr_occur++;
            }
            i++ ;
        }
    }
    printf("\n le nombre de sous chaines de longueur max est : %d\n",nbr_occur);
}

void exo_23_v2(char *str)
{
    int i = 0, nbr_occur = 0 , nbc = strlen(str)-1;

    for(i = 0 ; i < nbc ; i++)
    {
        if (str[i] == str[i+1])
        {
            printf("\n %s",str+i);
            nbr_occur++;
        }
    }
    printf("\n le nombre de sous chaines de longueur max est : %d\n",nbr_occur);
}
```

52. Trouver dans un tableau de caractères toutes les sous-chaînes de caractères dont la première lettre et la seconde sont identiques. Affichez le nombre de sous-chaînes trouvées.

```c
void exo_24(char *str)
{
    int i = 0, nbr_occur = 0 , nbc = strlen(str)-1;
    int nb_sc;
    for(i = 0 ; i < nbc ; i++)
    {
        if (str[i] == str[i+1])
        {
            nb_sc = strlen(str+i)-1;
            nbr_occur += nb_sc;
        }
    }
    printf("\n le nombre de sous chaines est : %d\n",nbr_occur);
}
```

53. Ecrire la fonction « **PositionChaine »** permettant de rechercher la position de la première occurrence d'une chaine « **t** » à l'intérieure d'une autre chaine « **s** ». Si la chaine « **t** » n'existe pas à l'intérieur de la chaine « **s** », la fonction renvoie la valeur **-1**.

Prototype : **int** PositionChaine (**char** ***** s , **char** ***** t)

```c
int PositionChaine(char *ch1, char *ch2)
{
    int i, j, k, taille1 = strlen(ch1), taille2 = strlen(ch2);
    for(i = 0; i < taille1 ; i++)
    {
        k = i;
        j = 0;
        while(k < taille1 && j < taille2 && ch1[k] == ch2[j])
        {
            k++;
            j++;
        }
        if(j == taille2)
            return i;
    }
    return -1;
}
```

54. Ecrire la fonction « **PositionChaineIndice »** permettant de rechercher la première position d'une chaine « **t** » à l'intérieure d'une autre chaine « **s** » à partir d'un indice donné. Si la chaine « **t** » n'existe pas à l'intérieur de la chaine « **s** » à partir de cet indice, la fonction renvoie la valeur -**1**.

Prototype : **int** PositionChaineIndice (**char** ***** s , **char** ***** t , **unsigned int** pos)

```c
int PositionChaineIndice(char *ch1, char *ch2, unsigned int ind)
{
    int i, j, k, taille1 = strlen(ch1), taille2 = strlen(ch2);
    for(i = ind ; i < taille1 ; i++)
    {
        k = i;
        j = 0;
        while(k < taille1 && j < taille2 && ch1[k] == ch2[j])
        {
            k++;
            j++;
        }
        if(j == taille2)
            return i;
    }
    return -1;
}
```

55. Ecrire une procédure « **PositionsChaine »** permettant de rechercher et d'afficher toutes les positions d'une chaine « **t** » à l'intérieure d'une autre chaine « **s** ».

    Prototype : **void** PositionsChaine (**char** * s , **char** * t)

```c
void PositionsChaine(char  * ch1, char * ch2)
{
    int pos = -1 ;
    while((pos = PositionChaineIndice(ch1,ch2, pos+1)) != -1)
    {
        printf(" \n position : %d ",pos);
    }
}
```

56. Ecrire la fonction « ListePositionsChaine » permettant retourner un tableau contenant toutes les positions des occurrences de la chaine « t » à l'intérieure de la chaine « s ».

    Prototype : **int** * ListePositionsChaine (**char** * s , **char** * t , **int** * nbe)

```c
int* ListePositionsChaine(char  * ch1, char * ch2, int *nbe)
{
    int pas = 1, pos = -pas , *T = NULL ;

    *nbe = 0 ;
    while((pos = PositionChaineIndice(ch1, ch2, pos + pas)) != -1)
    {
        T = realloc(T,(*nbe+1)*sizeof(int));
        T[(*nbe)++] = pos;
    }
    return T;
}
```

57. Adapter la fonction « ListePositionsChaine » (exo 28) pour retourner un tableau contenant toutes les positions du mot « t » ou de l'occurrence « t » à l'intérieure de la chaine « s ».

Prototype : **int *** ListePositionsChaine (**char *** s , **char *** t , **int *** nb_occur , **char** opt)

```
int* ListePositionsChaine(char  * ch1, char * ch2, int *nbe, char opt)
{
    int pas = 1, pos = -pas , *T = NULL ;
    if((opt == 'w') || (opt == 'W'))
    {
        pas = strlen(ch2) ;
        pos = - pas ;
    }
    *nbe = 0 ;
    while((pos = PositionChaineIndice(ch1, ch2, pos + pas)) != -1)
    {
        T = realloc(T,(*nbe+1)*sizeof(int));
        T[(*nbe)++] = pos;
    }
    return T;
}
```

58. Ecrire la fonction « SaisirChaine » permettant de saisir une chaine de caractères au clavier et lui allouant la taille mémoire exacte sans pour autant connaitre sa taille à l'avance.

Prototype : **char *** SaisirChaine ()

```
char * SaisirChaine()
{
    unsigned int c, nbc = 0 ;
    char * ch = NULL ;

    while((c = getchar()) != '\n')
    {
        ch = realloc(ch, (nbc+1)*sizeof(char));
        ch[nbc++] = c ;
    }
    ch = realloc(ch, (nbc+1)*sizeof(char));
    ch[nbc] = '\0' ;

    return ch ;
}
```