Exercice 1 Déclaration d'une chaîne de caractères ** Parmi les déclarations de variables suivantes, quelles sont celles qui sont correctes? Dans ce cas, quel est l'espace pris en mémoire par la variable?*

1. `char a[] = "un\ndeux\ntrois\n";`
2. `char b[10] = "un deux trois";`
3. `char c[] = 'abcdefg';`
4. `char d[10] = 'x';`
5. `char e[5] = "cinq";`
6. `char f[] = "Cette " "phrase" " est rompue ";`
7. `char g[2] = {'a','\0'};`
8. `char h[4] = {'a', 'b', 'c'};`
9. `char i[4] = "'o'";`

Solution *1*

1. *Déclaration correcte, espace: 15 octets*
2. *Déclaration incorrecte: la chaîne d'initialisation dépasse le bloc de mémoire réservé.*
   Correction: `char b[14]="un deux trois";` *ou mieux:* `char b[]= "un deux trois";` *, espace: 14 octets*
3. *Déclaration incorrecte: les symboles ' et " encadrent des caractères; pour initialiser avec une chane de caractères, il faut utiliser les guillemets (ou indiquer une liste de caractres).*
   Correction: `char c[] = "abcdefg";`*, espace: 8 octets*
4. *Déclaration incorrecte: Il faut utiliser une liste de caractères ou une chaîne pour l'initialisation.*
   Correction: `char d[10] = {'x', '\0'};` *ou mieux :* `char d[10] = "x";`*, espace: 2 octets*
5. *Déclaration correcte, espace: 5 octets*
6. *Déclaration correcte, espace: 23 octets, mais à éviter pour son manque de clareté*
7. *Déclaration correcte, espace: 2 octets*
8. *Déclaration incorrecte: Dans une liste de caractères, il faut aussi indiquer le symbole de fin de chaîne.*
   Correction: `char h[4] = {'a', 'b', 'c', '\0'};`*, espace: 4 octets*
9. *Déclaration correcte, mais d'une chaîne contenant les caractères* `'\'','o','\''et'\0'`*, espace: 4 octets*

Exercice *2* Opérations sur les chaînes de caractères **

1. *Écrire un code qui permet de calculer la longueur d'une chaîne de caractères donnée.*
2. *Écrire un code qui permet de copier une chaîne de caractères dans une autre (pourquoi ne peut-on pas utiliser l'opération* `s1=s2;` *?).*
3. *Écrire un code qui permet de supprimer les deux dernières lettres d'une chaîne de caractères.*
4. *La* **concaténation** *des deux chaînes de caractères S_1 et S_2 (généralement notée : $S_1 \# S_2 = S_{cat}$) revient à construire une chaîne de caractères S_{cat} contenant tous les caractères de S_1 puis tous les caractères de S_2. Écrire un programme réalisant la concaténation de deux chaînes de caractères données par l'utilisateur.*

Solution *2*

```
#include <stdio.h>
#include <string.h>
#define _TAILLE1 256
#define _TAILLE2 256
int main()
    {
/**** Dclarations */
    char phrase[_TAILLE1];/* Phrase a analyser */
    char phrase2[_TAILLE2];/* Phrase de destination */
    char phrasecat[_TAILLE2+_TAILLE2-1];/* Phrase concatenee */
    int L;/* longueur de la phrase */
    char c;/* caractere tampon */
    int i,j;/* compteurs*/
    printf("Tapez une phrase [%d caracteres maximum]\n",_TAILLE1-1);
/***** Lecture de la chaine */
    // scanf("%s",phrase);
    //!!!l'utilisation de scanf elimine les espaces qu'il prend pour une fin de chaine !!!!
    i=0;
    do {
        c=getchar();
        if (c!='\n')
            {
            phrase[i]=c; /* On copie le caractere */
            }
        else
            {
            phrase[i]='\0'; /* On remplace le retour ligne par une fin chaine */
            }
        i++;
        }
    while ((c!='\n')&&(i<_TAILLE1)); /* ne pas oublier le ; !!!!!*/
/***** Comptage du nombre de caracteres */
    L=0;
    while(phrase[L]!='\0'){L++;}
    /* Affichage du resultat */
    printf("Cette phrase est de longueur %d\n",L);
    i=0;/* reinitialisation du compteur */

/******   Copie de la chaine */
    int depassement_tableau=0;/* booleen de depassement de tableau*/
while(phrase[i]!='\0'&&!depassement_tableau)
        {
        if (i>(_TAILLE2-1))
            {
            depassement_tableau=1;
            }
else
    {
        phrase2[i]=phrase[i];i++;
    }
        }
if (depassement_tableau)
    {
             printf("Erreur : la chaine de destination est trop courte\n");
    }
   phrase2[i]='\0';
    /* Affichage du resultat */
    printf("Avez vous bien tape %s\n",phrase2);
/**** Supression des deux derniers caractres de phrase 2 */
    phrase2[L-2]='\0';
    printf("Et sans les derniers caracteres %s\n",phrase2);
/**** Concatenation des deux chaines ***/
    //!!! en general, ne pas oublier de verifier les tailles des chaines
    j=0;i=0;
    while(phrase[i]!='\0')
        {phrasecat[j]=phrase[i];i++;j++;}/* Ecriture de la premiere phrase */
```

```
    i=0;
    while(phrase2[i]!='\0')
        {phrasecat[j]=phrase2[i];i++;j++;}/* Puis de la seconde phrase */
    phrasecat[j]='\0';/* et fin de chaine */
     printf("Les deux phrases concatenees font: %s\n",phrasecat);

return 1;
}
```

Exercice *3* Ordre Lexicographique ***

De même que l'on peut comparer deux entiers, on peut comparer deux chaînes de caractères par une relation d'ordre appellée ordre lexicographique (ou ordre du dictionnaire).

L'ordre entre les caractère est l'ordre dans la table ASCII. Ensuite, pour comparer deux chaînes de caractères $a = (a_1 \dots a_n)$ *et* $b = (b_1 \dots b_m)$*, on parcourt* a *de gauche à droite, et on compare le caractère* (a_i) *au caractère* (b_i)*. Le premier caractère* (a_k) *rencontré différent de* (b_k) *définit l'ordre : si* $(a_k) < (b_k)$ *alors* $a < b$ *sinon* $a > b$*. Si on ne rencontre aucun caractère différent, la chaîne la plus courte est la plus petite.*

Exemples : `piano<piscine ; cabine<car ; Mias 1<mias 1`.

Ecrire un programme qui compare deux chaînes de caractères fournies par l'utilisateur.

Solution *3*

```
#include <stdio.h>
#define _TAILLE 50
int main()
    {
    /* Dclarations */
    char CH1[_TAILLE], CH2[_TAILLE]; /* chanes  comparer */
    int I;                  /* indice courant     */
    char c;                 /* caractere tampon */
    /* Saisie des donnes */
    printf("Entrez la premire chane  comparer : ");
    I=0;
    do {
        c=getchar();
        if (c!='\n')
            {
            CH1[I]=c; /* On copie le caractere */
            }
        else
            {
            CH1[I]='\0'; /* On remplace le retour ligne par une fin chaine */
            }
        I++;
        }
    while ((c!='\n')&&(I<_TAILLE)); /* ne pas oublier le ; !!!!!*/
    printf("Entrez la deuxime chane  comparer : ");
    I=0;
    do {
        c=getchar();
        if (c!='\n')
            {
            CH2[I]=c; /* On copie le caractere */
            }
        else
            {
            CH2[I]='\0'; /* On remplace le retour ligne par une fin chaine */
            }
        I++;
        }
    while ((c!='\n')&&(I<_TAILLE)); /* ne pas oublier le ; !!!!!*/
    /* Chercher la premire position o */
    /* CH1 et CH2 se distinguent. */
    I=0;
    while( (CH1[I]==CH2[I]) && (CH1[I]!='\0') && (CH2[I]!='\0'))
```

```
        {
        I++;
        }
    /* Comparer le premier lment qui */
    /* distingue CH1 et CH2. */
    if (CH1[I]==CH2[I])
        printf("\"%s\" est gal  \"%s\"\n", CH1, CH2);
    else if (CH1[I]<CH2[I])
        printf("\"%s\" prcde \"%s\"\n", CH1, CH2);
    else
        printf("\"%s\" prcde \"%s\"\n", CH2, CH1);
    return 1;
    }
```

Exercice *4* Tri d'un tableau **

Le tri d'un tableau selon ses valeurs croissantes est un algorithme souvent utilisé en informatique. Dans cet exercice, on va coder une algorithme de tri standard, mais on verra en T.P. une méthode plus raffinée.

On suppose que le tableau à classer A est un tableau de variables dont le type permet de définir un ordre (`int,float, char, char[],...`*).*

Tri par sélection du minimum : Parcourir le tableau de gauche droite à l'aide de l'indice I. Pour chaque élément $A[I]$ du tableau, déterminer la position $PMIN$ du (premier) minimum à droite de $A[I]$ et échanger $A[I]$ et $A[PMIN]$.

Écrire en C un algorithme de tri par sélection du minimum sur un tableau d'entiers.

Solution *4*
```
#include <stdio.h>
 #define _TAILLE_MAX 50
int  main()
    {
    /* Dclarations */
    int A[_TAILLE_MAX]; /* tableau donn */
    int N;     /* dimension     */
    int I;     /* rang  partir duquel A n'est pas tri */
    int J;     /* indice courant      */
    int SWAP;  /* pour la permutation */
    int PMIN;  /* indique la position de l'lment */
    /* minimal  droite de A[I]         */
    /* Saisie des donnes */
    printf("Dimension du tableau (max.%d) : ",_TAILLE_MAX);
    scanf("%d", &N );
    for (J=0; J<N; J++)
        {
        printf("Elment %d : ", J);
        scanf("%d", &A[J]);
        }
    /* Affichage du tableau */
    printf("Tableau donn :\n");
    for (J=0; J<N; J++)
        printf("%d ", A[J]);
    printf("\n");
    /* Tri du tableau par slection directe du minimum. */
    for (I=0; I<N-1; I++)
        {
        /* Recherche du maximum  droite de A[I] */
        PMIN=I;
        for (J=I+1; J<N; J++)
            if (A[J]<A[PMIN]) PMIN=J;
            /* Echange de A[I] avec le minimum */
            SWAP=A[I];
            A[I]=A[PMIN];
            A[PMIN]=SWAP;
        }
    /* Edition du rsultat */
    printf("Tableau tri :\n");
    for (J=0; J<N; J++)
        printf("%d ", A[J]);
```

```
    printf("\n");
    return 1;
    }
```

Exercice *5* Triangle de Pascal $**$

Ecrire un programme qui construit le triangle de Pascal de degré N et le mémorise dans une matrice carrée P de dimension N+1. Dans le triangle de Pascal, le coefficient $P_{n,j}$ *vaut* $C_n^j = \frac{n!}{j!(n-j)!}$.

On utilisera la relation de récurrence suivante : $P_{n,j} = P_{n-1,j-1} + P_{n-1,j}$*, en notant que* $P_{n,n} = P_{0,0} = 1$.

Exemple: Triangle de Pascal de degré 4

n=0 : 1

n=1 : 1 1

n=2 : 1 2 1

n=3 : 1 3 3 1

n=4 : 1 4 6 4 1

Solution *5*

```
#include <stdio.h>
#define _TAILLE_MAX 13
int main()
    {
    /* Dclarations */
    int P[_TAILLE_MAX+1][_TAILLE_MAX+1]; /* matrice rsultat  */
    int N;         /* degr du triangle */
    int I, J;      /* indices courants  */
    /* Saisie des donnes */
    do {
        printf("Entrez le degr N du triangle (max.%d) : ",_TAILLE_MAX);
        scanf("%d", &N);
        } while (N>_TAILLE_MAX||N<0);
    /* Construction des lignes 0  N du triangle: */
    /* Calcul des composantes du triangle jusqu' */
    /* la diagonale principale. */
    for (I=0; I<=N; I++)
        {
        P[I][I]=1;
        P[I][0]=1;
        for (J=1; J<I; J++)
            P[I][J] = P[I-1][J] + P[I-1][J-1];
        }
    /* Edition du rsultat */
    printf("Triangle de Pascal de degr %d :\n", N);
    for (I=0; I<=N; I++)
        {
        printf(" N=%2d", I);
        for (J=0; J<=I; J++)
            if (P[I][J])
                printf("%5d", P[I][J]);
            printf("\n");
        }
    return 1;
    }
```

Exercice *6* Structures $*$

Construire une structure pour stocker des données d'horaires (heures, minutes, secondes). Écrire un programme calculant la différence entre deux horaires en secondes.

Solution *6*

```
#include <stdio.h>
typedef struct
    {
    unsigned short int  heure;
    unsigned short int minute;
```

```
    unsigned short int seconde;
    } horaire;
main()
    {
/***Declarations */
    short int tamp;/* variable tampon */
    //!!!!  Si le tampon est declare comme unsigned, on ne pourra plus verifier
    //      si l'utilisateur rentre une valeur negative
    horaire hor1,hor2;/* les deux horaires */
    int diff; /* difference */
/**** Premier horaire */
    printf("Entrez un horaire\n");
    do
        {
    printf("Heures:");
        scanf( "%d",&tamp);
       }
    while((tamp>23)||((int)tamp<0));
    hor1.heure=(unsigned short) tamp;
    do
        {
    printf("Minutes:");
        scanf( "%d",&tamp);
       }
    while((tamp>59)||((int)tamp<0));
    hor1.minute=(unsigned short) tamp;
    do
        {
    printf("Secondes:");
        scanf( "%d",&tamp);
       }
    while((tamp>59)||((int)tamp<0));
    hor1.seconde=(unsigned short) tamp;
/**** Second horaire */
    printf("Entrez un second horaire\n");
    do
        {
    printf("Heures:");
        scanf( "%d",&tamp);
       }
    while((tamp>23)||((int)tamp<0));
    hor2.heure=(unsigned short) tamp;
    do
        {
    printf("Minutes:");
        scanf( "%d",&tamp);
       }
    while((tamp>59)||((int)tamp<0));
    hor2.minute=(unsigned short) tamp;
    do
        {
    printf("Secondes:");
        scanf( "%d",&tamp);
       }
    while((tamp>59)||((int)tamp<0));
    hor2.seconde=(unsigned short) tamp;
/*** Calcul de la difference */
    diff=((hor1.heure-hor2.heure)*3600+(hor1.minute-hor2.minute)*60
        +(hor1.seconde-hor2.seconde));
    printf("Entre %huh%hu'%hu'' et %huh%hu'%hu'', il y a %d secondes",
        hor1.heure,hor1.minute,hor1.seconde,hor2.heure,hor2.minute,hor1.seconde,diff);
    }
```

Exercice *7* Implémentation d'une sructure de pile **

Créer un structure nommée `Pile` *contenant deux champs : un tableau de taille fixée correspondant aux données dans la pile et un entier correspondant à l'indice du sommet de la pile. Coder les différentes actions sur la pile suivantes : initialiser*

la pile, tester si la pile est vide ou pleine, mettre un nouvel élément dans la pile, consulter le sommet de la pile, retirer un élément de la pile, compter le nombre d'éléments empilés.

Utiliser cette structure pour coder un algorithme qui vérifie si un mot donné est un palindrome (un mot qui se lit de la même manière dans les deux sens).

Algorithme :
Pile P1,P2,P3;
Remplir P1 et P2 avec les caractères de la chaîne (donnés par un utilisateur).
Tant que (P1 non vide)
Dépiler le contenu de P1 dans P3
Tant que (P3 non vide)
Dépiler le contenu de P2 et P3 et comparer les sommets
S'arrêter dès que deux sommets sont distincts

Solution 7

```
#include <stdio.h>
#define TAILLE 500

typedef struct {
char elements[TAILLE];
int haut; } PILE;

int main()
{
PILE p;char e='e';

/* INIT PILE */
p.haut = -1;

/* PUSH */
if (p.haut!=(TAILLE-1)) //SI PILE !PLEINE
{
p.haut++;
p.elements[p.haut] = e;

}
/* On aurait pu l'ecrire :
p.elements[++p.haut] = e; */

/* POP */
if (p.haut!=-1)   //SI PILE !VIDE
{
e=p.elements[p.haut];
p.haut--;
}

/* On aurait pu l'ecrire :
e=p.elements[p->haut--];*/


/*Le bout de code suivant permet de tester
si la squence de caractres donne en entre est un palindrome. */
PILE p1, p2, p3;

// REMISE A ZERO DES PILES
p1.haut=-1;p2.haut=-1;p3.haut=-1;
int c, c1, c2;
int palindrome=1;//booleen

printf("Tapez une phrase");

while ((c = getchar()) != '\n')
{
if (p1.haut!=(TAILLE-1)) //SI PILE !PLEINE
{
p1.haut++;
p1.elements[p1.haut] = c;
```

```
}
if (p2.haut!=(TAILLE-1)) //SI PILE !PLEINE
{
p2.haut++;
p2.elements[p2.haut] = c;

}
}


//printf("%d",p2.elements[p.haut]);


while (p1.haut!=-1)
{ // renverse le contenu de p1 dans p3

if (p1.haut!=-1)   //SI PILE !VIDE
{
c=p1.elements[p1.haut];
p1.haut--;
}

if (p3.haut!=(TAILLE-1)) //SI PILE !PLEINE
{
p3.haut++;
p3.elements[p3.haut] = c;

}

}

while ((p3.haut!=-1)&&(palindrome==1))
{ // test la squence originale (dans p2)
if (p2.haut!=-1)   //SI PILE !VIDE
{
c1=p2.elements[p2.haut];
p2.haut--;
}

// avec la squence inverse (dans p3)
if (p3.haut!=-1)   //SI PILE !VIDE
{
c2=p3.elements[p3.haut];
p3.haut--;
}

if (c1 !=c2)
palindrome=0;
}

if (palindrome)
{
printf ("le mot est un palindrome\n");
}
else
{
printf ("le mot n'est pas une palindrome\n");
}
return 1;
}
```

Exercice *8* Structures et type booléen *

Le C ne connait pas la type boolén auquel il préfère les entiers 0 *et* 1. *Comment construire un type booléen pouvant prendre les valeurs 'VRAI' ou 'FAUX' qui soit compatible avec les structures du C (if, while, etc...).*

Solution *8* *Tout d'abord, noter que le type* `bool` *a été introduit en C++, ce qui est la meilleure solution possible. Sous Visual, on ne peut donc pas*

redéfinir le type `bool`. *Proposer 3 types d'implémentation du type* `BOOL` *en utilisant* `typedef`, `define` *et* `enum`.

1. ```
   #define BOOL int
   #define VRAI 1
   #define FAUX 0
   ```

   - *Le champ des valeurs possibles pour* `BOOL` *n'est pas restreint à* `VRAI` *et* `FAUX`.
   - *Le type* `BOOL` *ne peut pas être distingué du type* `int` *(dans la surcharge de fonctions par exemple).*
   - *Comme pour toute constante macro , risque d'écrasement, de redéfinition puisqu'elle ne sont pas compilées...*

2. ```
   enum BOOL {FAUX,VRAI};
   ```

   Cette solution est relativement bonne dans la mesure ou elle palie à tous les défauts de la première option. Elle pose tout de même un problème majeur : on ne peut pas convertir un `int` *en* `BOOL`.

   ```
   BOOL b;
   b= (i==j);//erreur de compilation
   ```

   Par contre, la conversion d'un `BOOL` *en* `int` *est automatique et permet d'utiliser les structures de contrôle.*

   ```
   BOOL b=VRAI;
   if (b){printf("b est VRAI");}
   else {printf("b est FAUX");}
   ```