Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Soit les déclarations suivantes :

```
int n = 10 , p = 4 ;
long q = 2 ;
float x = 1.75 ;
```

Donner le type et la valeur de chacune des expressions suivantes :

a) `n + q`
b) `n + x`
c) `n % p +q`
d) `n < p`
e) `n >= p`
f) `n > q`
g) `q + 3 * (n > p)`
h) `q && n`
i) `(q-2) && (n-10)`
j) `x * (q==2)`
k) `x *(q=5)`

2) Écrire plus simplement l'instruction suivante :

```
z = (a>b ? a : b) + (a <= b ? a : b) ;
```

3) `n` étant de type `int`, écrire une expression qui prend la valeur :

`-1` si `n` est négatif,

`0` si `n` est nul,

`1` si `n` est positif.

4) Quels résultats fournit le programme suivant ?

```
#include <stdio.h>
main()
{
   int n=10, p=5, q=10, r ;
   r = n == (p = q) ;
   printf ("A : n = %d  p = %d  q = %d  r = %d\n",
n, p, q, r) ;
   n = p = q = 5 ;
   n += p += q ;
   printf ("B : n = %d  p = %d  q = %d\n", n, p, q)
;
   q = n < p ? n++ : p++ ;
   printf ("C : n = %d  p = %d  q = %d\n", n, p, q)
;
   q = n > p ? n++ : p++ ;
   printf ("D : n = %d  p = %d  q = %d\n", n, p, q)
;
}
```

Tous ces exercices sont corrigés en fin de volume.

1) Quels seront les résultats fournis par ce programme ?

```
#include <stdio.h>
main ()
{   int n = 543 ;
    int p = 5 ;
    float x = 34.5678;
    printf ("A : %d %f\n", n, x) ;
    printf ("B : %4d %10f\n", n, x) ;
    printf ("C : %2d %3f\n", n, x) ;
    printf ("D : %10.3f %10.3e\n", x, x) ;
    printf ("E : %*d\n", p, n) ;
    printf ("F : %*.*f\n", 12, 5, x) ;
}
```

2) Quelles seront les valeurs lues dans les variables *n* et *p* (de type *int*), par l'instruction suivante ?

```
scanf ("%4d %2d", &n, &p) ;
```

lorsqu'on lui fournit les données suivantes (le symbole ^ représente un espace et le symbole @ représente une fin de ligne, c'est-à-dire une validation) ?

a) 12^45@
b) 123456@
c) 123456^7@
d) 1^458@
e) ^^^4567^^8912@

Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Soit le petit programme suivant :

```
#include <stdio.h>
main()
{
   int i, n, som ;
   som = 0 ;
   for (i=0 ; i<4 ; i++)
      { printf ("donnez un entier ") ;
        scanf ("%d", &n) ;
        som += n ;
      }
   printf ("Somme : %d\n", som) ;
}
```

Écrire un programme réalisant exactement la même chose, en employant, à la place de l'instruction `for` :

- une instruction `while`,
- une instruction `do... while`.

2) Calculer la moyenne de notes fournies au clavier avec un dialogue de ce type :

```
note 1 : 12
note 2 : 15.25
```

```
note 3 : 13.5
note 4 : 8.75
note 5 : -1
moyenne de ces 4 notes : 12.37
```

Le nombre de notes n'est pas connu a priori et l'utilisateur peut en fournir autant qu'il le désire. Pour signaler qu'il a terminé, on convient qu'il fournira une note fictive négative. Celle-ci ne devra naturellement pas être prise en compte dans le calcul de la moyenne.

3) Afficher un triangle rempli d'étoiles, s'étendant sur un nombre de lignes fourni en donnée et se présentant comme dans cet exemple :

```
*
**
***
****
*****
```

4) Déterminer si un nombre entier fourni en donnée est premier ou non.

5) Écrire un programme qui détermine la *n*-ième valeur u_n (*n* étant fourni en donnée) de la « suite de Fibonacci » définie comme suit :

$u_1 = 1$

$u_2 = 1$

$u_n = u_{n-1} + u_{n-2}$ pour $n>2$

6) Écrire un programme qui affiche la table de multiplication des nombres de 1 à 10, sous la forme suivante :

```
      I   1   2   3   4   5   6   7   8   9  10
---------------------------------------------------
   1  I   1   2   3   4   5   6   7   8   9  10
   2  I   2   4   6   8  10  12  14  16  18  20
   3  I   3   6   9  12  15  18  21  24  27  30
   4  I   4   8  12  16  20  24  28  32  36  40
   5  I   5  10  15  20  25  30  35  40  45  50
   6  I   6  12  18  24  30  36  42  48  54  60
   7  I   7  14  21  28  35  42  49  56  63  70
   8  I   8  16  24  32  40  48  56  64  72  80
   9  I   9  18  27  36  45  54  63  72  81  90
  10  I  10  20  30  40  50  60  70  80  90 100
```

Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Écrire :

- une fonction, nommée `f1`, se contentant d'afficher `"bonjour"` (elle ne possèdera aucun argument ni valeur de retour),
- une fonction, nommée `f2`, qui affiche `"bonjour"` un nombre de fois égal à la valeur reçue en argument (`int`) et qui ne renvoie aucune valeur,
- une fonction, nommée `f3`, qui fait la même chose que `f2`, mais qui, de plus, renvoie la valeur (`int`) `0`.

Écrire un petit programme appelant successivement chacune de ces trois fonctions, après les avoir convenablement déclarées sous forme d'un prototype.

2) Qu'affiche le programme suivant ?

```
int n=5 ;
main()
{
   void fct (int p) ;
   int n=3 ;
   fct(n) ;
```

```
}
void fct(int p)
{
   printf("%d %d", n, p) ;
}
```

3) Écrire une fonction qui se contente de comptabiliser le nombre de fois où elle a été appelée en affichant seulement un message de temps en temps, à savoir :

- au premier appel : *** appel 1 fois ***
- au dixième appel : *** appel 10 fois ***
- au centième appel : *** appel 100 fois ***
- et ainsi de suite pour le millième, le dix millième appel...
- On supposera que le nombre maximal d'appels ne peut dépasser la capacité d'un `long`.

4) Écrire une fonction récursive calculant la valeur de la « fonction d'Ackermann » `A` définie pour `m>0` et `n>0` par :

```
A(m,n) = A(m-1,A(m,n-1))    pour m>0 et n>0
A(0,n) = n+1    pour n>0
A(m,0) = A(m-1,1)    pour m>0
```

Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Écrire, de deux façons différentes, un programme qui lit 10 nombres entiers dans un tableau avant d'en rechercher le plus grand et le plus petit :

- en utilisant uniquement le « formalisme tableau »,
- en utilisant le « formalisme pointeur », chaque fois que cela est possible.

2) Écrire une fonction qui ne renvoie aucune valeur et qui détermine la valeur maximale et la valeur minimale d'un tableau d'entiers (à un indice) de taille quelconque. Il faudra donc prévoir 4 arguments : le tableau, sa dimension, le maximum et le minimum.

Écrire un petit programme d'essai.

3) Écrire une fonction permettant de trier par ordre croissant les valeurs entières d'un tableau de taille quelconque (transmise en argument). Le tri pourra se faire par réarrangement des valeurs au sein du tableau lui-même.

4) Écrire une fonction calculant la somme de deux matrices dont les éléments sont de type `double`. Les adresses des trois matrices et leurs dimensions (communes) seront transmises en argument.

Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Écrire un programme déterminant le nombre de lettres « e » (minuscules) présentes dans un texte de moins d'une ligne (supposée ne pas dépasser 132 caractères) fourni au clavier.

2) Écrire un programme qui supprime toutes les lettres « e » (minuscules) d'un texte de moins d'une ligne (supposée ne pas dépasser 132 caractères) fourni au clavier. Le texte ainsi modifié sera créé, en mémoire, à la place de l'ancien.

3) Écrire un programme qui lit au clavier un mot (d'au plus 30 caractères) et qui l'affiche à l'envers.

4) Écrire un programme qui lit un verbe du premier groupe et qui en affiche la conjugaison au présent de l'indicatif, sous la forme :

```
je chante
tu chantes
il chante
nous chantons
vous chantez
ils chantent
```

Le programme devra vérifier que le mot fourni se termine bien par « er ». On supposera qu'il ne peut comporter plus de 26 lettres et qu'il s'agit d'un verbe régulier. Autrement dit, on admettra que l'utilisateur ne fournira pas un verbe tel que « manger » (le programme afficherait alors : « nous mangons »).

Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Écrire un programme qui :

- lit au clavier des informations dans un tableau de structures du type `point` défini comme suit :

```
struct point { int num ;
               float x ;
               float y ;
             }
```

Le nombre d'éléments du tableau sera fixé par une instruction `#define`.

- affiche à l'écran l'ensemble des informations précédentes.

2) Réaliser la même chose que dans l'exercice précédent, mais en prévoyant, cette fois, une fonction pour la lecture des informations et une fonction pour l'affichage.

Exercices

Tous ces exercices sont corrigés en fin de volume.

1) Écrire un programme permettant d'afficher le contenu d'un fichier texte en numérotant les lignes. Ces lignes ne devront jamais comporter plus de 80 caractères.

2) Écrire un programme permettant de créer séquentiellement un fichier « répertoire » comportant pour chaque personne :

- nom (20 caractères maximum) ;
- prénom (15 caractères maximum) ;

Les informations relatives aux différentes personnes seront lues au clavier.

3) Écrire un programme permettant, à partir du fichier créé par l'exercice précédent, de retrouver les informations correspondant à une personne de nom donné.

4) Écrire un programme permettant, à partir du fichier créé dans l'exercice 2, de retrouver les informations relatives à une personne de rang donné (par accès direct).

ÉNONCÉS DES EXERCICES ET DES PROBLÈMES

EXERCICES

Exercice 3.1 Afficher une liste chaînée ****

Écrire un algorithme pour afficher une liste chaînée d'entiers sous la forme suivante :
Si la liste est linéaire :

```
| Nom_liste: .->[elt1]->[elt2]->...->[eltn]->[eltlast] |
```

Si la liste est circulaire :

```
| Nom_liste: .->[elt1]->[elt2]->...->[eltn]->[eltlast] ...
```

Si certains maillons contigus sont doublement chaînés, afficher `]<->[` au lieu de `]->[`.

Apportez une attention particulière aux traitements spécifiques de début, milieu et fin de liste ainsi qu'aux cas spéciaux comme la liste vide ou singleton.

Exercice 3.2 Construire une liste circulaire ordonnée ***

Écrire un algorithme qui crée une liste circulaire ordonnée d'entiers à partir d'un tableau non ordonné d'entiers.

Exercice 3.3 Réaliser le chaînage arrière d'une liste doublement chaînée *

Concevoir un algorithme qui réalise le chaînage arrière d'une liste doublement chaînée dont seul le chaînage avant a été effectué.

Exercice 3.4 Inverser pile et file **

Concevoir deux algorithmes, qui créent respectivement :

- la file inverse d'une file ;
- la pile inverse d'une pile.

Ces deux algorithmes doivent restituer leur entrée inchangée.

Exercice 3.5 Simuler la récursivité à l'aide d'une pile *

Écrire un algorithme pour calculer la somme de 1 à n $\in n \in \mathbb{N}^*$ en simulant la récursivité à l'aide d'une pile.

Les en-têtes des opérations à utiliser pour cet exercice sont fournis :
FONCTION nouvellePile() : pile
FONCTION estPileVide(p : pile) : booléen
FONCTION empiler (val : T, *pp : pile)
FONCTION depiler (*pval : T, *pp : pile) : entier

Exercice 3.6 Insérer et supprimer dans une liste doublement chaînée **

Écrire un algorithme d'insertion dans une liste doublement chaînée.

Écrire un algorithme de suppression dans une liste doublement chaînée.

EXERCICES

Exercice 4.1 Traiter un arbre en post-ordre *

- a) Écrire un algorithme récursif de traitement en post-ordre d'un arbre binaire.
- b) Dérouler l'algorithme sur l'exemple suivant :

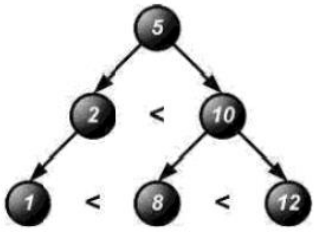

Figure 4.21

Exercice 4.2 Afficher le contenu des feuilles d'un arbre **

Écrire un algorithme permettant d'afficher tous les entiers enregistrés dans les feuilles d'un arbre binaire (en ignorant les entiers contenus dans tous les autres nœuds).

Exercice 4.3 Rechercher itérativement un élément dans un ABOH *

Écrire un algorithme itératif de recherche d'un élément dans un ABOH.

Exercice 4.4 Évaluer le caractère ABOH d'un AB ***

Écrire un algorithme permettant de déterminer si un arbre binaire donné est ou n'est pas un arbre ABOH.

Exercice 4.5 Mesurer la hauteur d'un ABOH **

Écrire un algorithme de calcul de la hauteur d'un ABOH.

Exercice 4.6 Évaluer l'équilibre d'un AB ***

Écrire un algorithme permettant de savoir si un arbre binaire est partiellement équilibré.

Exercice 4.7 Parcourir un AB en largeur et en profondeur ***

Écrire deux algorithmes, respectivement de parcours en largeur d'abord et en profondeur d'abord, pour afficher le contenu d'un arbre binaire.

Exercice 4.8 Effectuer un calcul complexe sur un arbre ****

Écrire un algorithme qui retourne la moyenne des éléments positifs et celle des éléments négatifs d'un arbre (0 est considéré ici comme un positif).

Exercice 4.9 Extraire une liste d'éléments d'un arbre ****

Écrire un algorithme qui retourne la liste chaînée des éléments d'un arbre d'entiers, qui ne sont pas divisibles par leur parent et que leur parent ne divise pas.

Citez au moins deux façons de construire des arbres pour lesquels cet algorithme retourne nécessairement la liste de tous ses éléments.

Exercice 4.10 Produire des coupes transversales d'un arbre ***

Coupe transversale d'un arbre sur un niveau donné, restituée sous forme de liste.

Écrire un algorithme qui retourne la coupe transversale.