Application

Ecrire un algorithme qui permet de tester la validité d'une date saisie sous la forme de jour, mois et année

Solution

```
ALGORITHME   Validité_Date
VAR
            J,M,A: entier
DEBUT
    Ecrire ("Donnez la date :")    Lire (J,M,A)
    Si ( (j<1 ou j>31) ou (M<1 ou M>12) ou ( A<0))
                Alors Ecrire(" date non valide ")
              Sinon
                    Selon M faire
                      1,3,5,7,8,10,12 : Ecrire (" date valide ")
                      4,6,9,11 :  Si (j=31) Alors Ecrire (" date non valide ")
                                  Sinon Ecrire (" date valide ")
                                  FinsSi
                       2 : Si ((j<29) ou ( j=29 et A mod 4 =0 et A mod 100 <>0))
                                  Alors Ecrire (" date valide ")
                                  Sinon Ecrie (" date non valide ")
                                FinSi
                    FinSelon
    FinSi
FIN
```

• Exemple

▫ Ecrire un algorithme qui permet de calculer

et afficher le PGCD de 2 entiers a et b

• Présentation informelle

Soient a, b deux entiers positifs (a>b).

Soient

q le quotient et

r le reste de la

division entière de a par b.

Alors pgcd(a,b) = pgcd(b,r)

Exemple

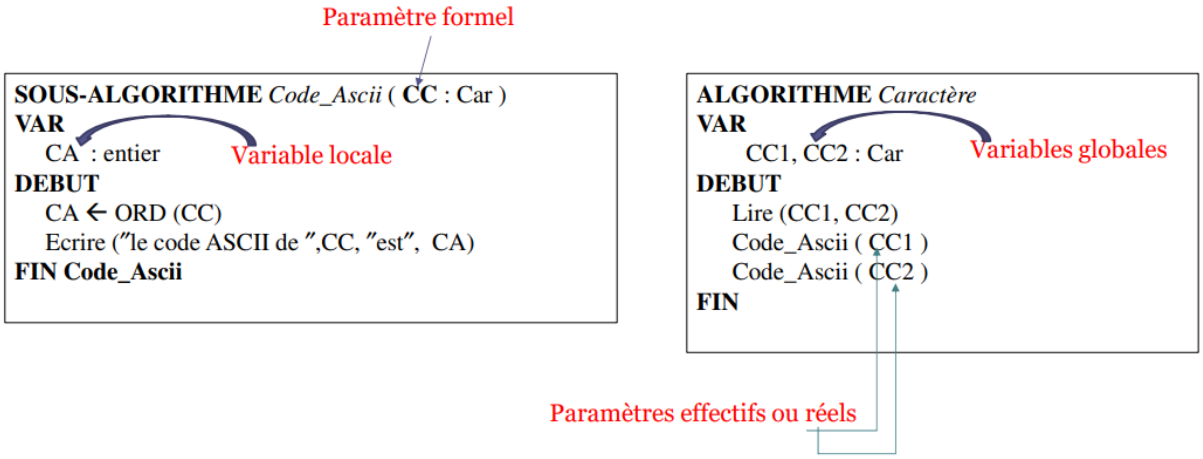
pgcd(105,30)=pgcd(30,15)=pgcd(15,0)=15

- **Solution**

```
ALGORITHME   PGCD
VAR
            a, b, P : entier
DEBUT
            Ecrire ("Donnez les 2 entiers :")
            Lire (a, b)
            // permutation si nécessiare
          si (a<b) alors
             a←a+b
                  b←a-b
                  a ← a-b
          finsi

            tant que ( b <> 0) faire
                              r ← a mod b
                              a ← b
                              b ← r
            Fin Faire
            p ←a
            Ecrire (" Le pgcd  est ", p )
FIN
```

Paramètres formels / Paramètres effectifs



- Exemple
  - calculer la combinaison P parmi N donnée par la formule $\frac{n!}{p!(n-p)!}$

```
FONCTION Fact ( X : entier ) : entier
VAR
      i , F : entier
DEBUT
      F ← 1
      POUR i de 2 à X FAIRE
            F ← F * i
      FIN FAIRE
Retourner F
FIN Fact
```

```
ALGORITHME Combinaison
VAR
      N,P,C: entier
DEBUT
      Lire (N,P)

      C ← Fact(N)/(Fact(P)*Fact (N-P))

      Ecrire (″le résultat est : ″, C )
FIN
```

- Exemple : Donnez la trace de l'algorithme X

```
ALGORITHME X
VAR
      N, P, Q : entier
DEBUT
      N ← 2
      P ← 2
      Q ← 2
      Calculer (N, P, Q)
      Ecrire (N, P, Q)
FIN
```

```
PROCEDURE Calculer (    ENTREE A : entier ;
                        SORTIE B : entier ;
                        E/S C : entier )
DEBUT
      A ← A-1
      B ← A + 5
      C ← C * 3
      Ecrire (A, B, C)

FIN Calculer
```

- Ecrire un sous algorithme permettant de déterminer le plus grand élément parmi les éléments d'un tableau T de N entiers, ainsi que son rang (position)

```
Procédure MAXIMUM  (Entrée T : tableau [1..30] d'entier, N : entier, Sortie max : entier, pos : entier)
   Var
    i : entier
   Début
// calcul du max  et de sa position
    pos ← 1
    pour i de 2 à N faire
      Si T [ i ] > T[pos] alors
        pos ← i
      FinSi
   FinFaire
max ← T [pos]
Fin MAXIMUM
```

- Ecrire un algorithme permettant de remplir une matrice
  - Ligne par ligne
  - Colonne par colonne

```
Algorithme Remplissage ligne_par_ligne
VAR
 M : tableau [1..100, 1..50] d'entier
 i,j , L, C : entier
Début
Répéter
    Ecrire(" donnez le nombre de lignes ")
    Lire(L)
Jusqu'à ( L>=1 et L<=100)
Répéter
    Ecrire(" donnez le nombre de colonnes ")
    Lire(C)
Jusqu'à ( C>=1 et C<=50)
```

```
// remplissage ligne par ligne
Pour i de 1 à L faire
  Pour j de 1 à C faire
    Ecrire(" donnez l'élément " , i , j)
    lire(M [i , j ])
  Fin faire
Fin faire
Fin
```

- Ecrire un algorithme permettant de remplir une matrice
  - Ligne par ligne
  - Colonne par colonne

```
Algorithme Remplissage colonne_par_colonne
VAR
 M : tableau [1..100, 1..50] d'entier
 i,j , L, C : entier
Début
Répéter
    Ecrire(" donnez le nombre de lignes ")
    Lire(L)
Jusqu'à ( L>=1 et L<=100)
Répéter
    Ecrire(" donnez le nombre de colonnes ")
    Lire(C)
Jusqu'à ( C>=1 et C<=50)
```

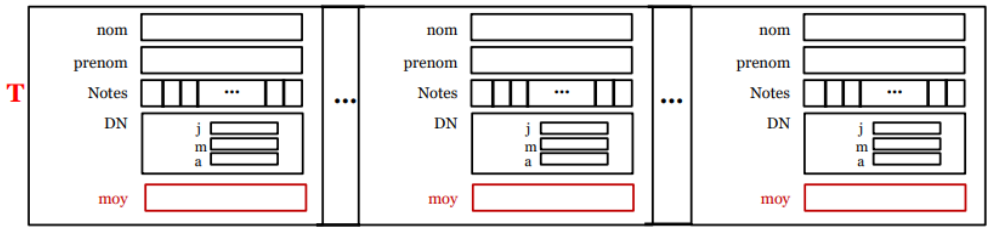
```
  // remplissage colonne par colonne
Pour j de 1 à C faire
  Pour i de 1 à L faire
    Ecrire(" donnez l'élément " , i , j)
    lire(M [i , j ])
  Fin faire
Fin faire
Fin
```

- Ecrire une fonction qui admet comme paramètres une matrice M de L x C réels et un indice de ligne i et qui retourne la somme des éléments de la i ème ligne de M

```
Fonction Somme_Ligne (M: tableau [1..20,1..30] de réel, C,  i : entier)  : réel
VAR
 j : entier
 S : réel
Début
    S ← 0
    pour j de 1 à C faire
        S ← S + M[i,j]
    Fin faire
retourner S
Fin Somme_Ligne
```

Tableaux de structures : Application

- Application 4 : Sachant qu'un étudiant est caractérisé par son nom, son prénom, ses notes dans 10 matières, sa moyenne générale et sa date de naissance, écrire un algorithme qui permet de
  - Remplir un tableau de N étudiants (N≤30) sachant que la moyenne doit être calculée à partir des moyennes des différentes matières (supposées de même coefficient)
  - Chercher le/les étudiants majeurs de la classes
  - Calculer la moyenne générale de la classe



Application

```
Algorithme Application4

TYPE
Date = Structure
        j,m,a : entier
      Fin Structure

Etudiant = Structure
        nom : chaine
        prenom : chaine
        DN : Date
        Notes : tableau [1..10] d'entier
        moy : réel
      Fin Structure
```

```
VAR
  T : Tableau [1..30] de Etudiant
  I,N : entier
  J,S : entier // somme des notes d'un étudiant
  max : reel  // meilleure moyenne
  SM : reel  // somme des moyennes des étudiants
        //pour le calcul de la moyenne générale de la classe
  moyG : reel // moyenne générale de la classe
```

```
Début
    // saisie de la taille de T
    Répéter
      ecrire(" donnez le nombre d'étudiants ")
      lire(N)
    Jusqu'à (N≥1 et N≤ 30)
    // remplissage de T
    Pour i de 1 à N faire
        ecrire(" donnez le nom et prénom " )
        lire(T[ i ].nom, T[i].prenom)
        ecrire(" donnez la date de naissance ")
        lire(T[ i ].DN.j, T[ i ].DN.m , T[ i ].DN.a)
        ecrire (" donnez les notes")
        S ← 0
        Pour j de 1 à 10 faire
            ecrire(" matiere n° ", j)
            lire(T[ i ].Notes [ j ])
            S ← S + T[ i ].Notes [ j ]
        Finfaire
      T[ i ].moy ← S /10
    Fin Faire
```

```
// calcul de la meilleur moyenne de la classe
//et calcul de la somme des moyennes
Max ← T[ 1 ]. moy
SM ← T[ 1 ]. moy
Pour i de 2 à N faire
  Si T[ i ]. moy  > max
        alors max ←T[ i ]. moy
   FinSi
   SM ←SM + T[ i ]. moy
FinFaire

//Affichage des nom et prénoms de meilleurs étudiants
ecrire(" les meilleurs étudiants sont " )
Pour i de 1 à N faire
        si T[ i ]. moy = max
            alors ecrire(T[ i]. nom, T[ i]. prenom)
        finSi
FinFaire
// calcul de la moyenne générale de la classe
  MG ← SM / N
ecrire(" la moyenne de la classe est ", MG)
Fin
```

Tri par sélection : Réalisation en C

Réalisation en C : Schéma 1

```
# define n 100
 int a[n] ; /* tableau à trier*/
 void tri_selection()
 {
                /* Partie déclarative : variables locales*/
                unsigned i ;
                /* varie entre 0 et n-2. Elle indique l'avancement dans le tri */

                 unsigned j ;
                /* varie entre i+1 et n-1. Elle permet de calculer m */

                unsigned m ;
                /*correspond à l'emplacement de l'élément le plus petit entre i
et n-1 */

                int t;
                /*utilisée pour l'échange entre a[i] et a[m]*/
```

```
/* partie exécutive */
for (i=0 ; i<n-1 ; i++)
                { /* recherche de  m */
                                m=i;
                                for(j=i+1 ; j<n ; j ++)
                                                if(a[j] < a[m])
                                                                m=j;
                  /* échange entre a[i] et a[m] */
                                t=a[i] ;
                                a[i]=a[m] ;
                                a[m]=t ;
                } /* fin for i */
}/* fin tri_selection */
```

Tester la procédure de tri

- résultat prévu à l'avance
- résultat observés suite à l'exécution du programme sur un jeu de données bien déterminées
- comparaison

```
#include <stdio.h>
# define n 5
 int a[n] ={18,14,10,8,4};
 void main()
 {
                tri_selection ();
                for(i=0 ;i<n ; i++)
   {
                                printf("%d", a[i]");
   }
 }
```

- Jeu de données
  - Données de test={18,14,10,8,4}
- Résultat prévu : 4 8 10 14 18
- Résultat observé
  - exécution du programme .....

Tri par insertion : Réalisation en C

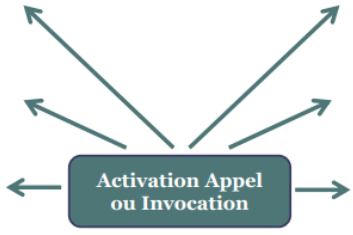
Réalisation en C

```
# define n 100
/* variables globales */
 int a[n] ;/*a : tableau à trier*/
 void tri_insertion()
{
        /* variables locales */
  unsigned i ;
        /* niveau d'avancement dans le tri */
  unsigned j ;
        /* pour le décalage */
   int v ;
        /*élément à insérer*/
/* partie exécutive */
for (i=1 ;i<n ;i++)
{
        /*insertion de a[i]*/
        v=a[i];
        j=i;
/* faire les décalages éventuels. Le nombre de décalages n'est pas connu a priori, on opte donc pour le schéma while plutôt que le schéma for */
        while ((j>0) && (a[j-1]>v))
        {
                a[j]=a[j-1];
                j--;
        /*passer à l'élément précédent*/
  } /* fin while */
/* à la sortie de la boucle  j<=0 || a[j-1] <= v */
a[j]=v ;
} /* fin for */
} /* fin tri_insertion */
```

Tri : Utilisation

Tri par insertion

```
void main()
{
        int a[5]={20,30,40,50,60};
        int b[5]={60,50,40,30,20};
        int c[8]={3,8,5,4,12,13,100,12};
        unsigned i;
        tri_insertion(a,5);
        for(i=0;i<5;i++)
                printf("%d\t"; a[i]);
        printf("\n");
        tri_insertion(b,5);
        for(i=0;i<5;i++)
                printf("%d\t"; b[i]);
        printf("\n");
        tri_insertion(c,8);
        for(i=0;i<8;i++)
                printf("%d\t"; c[i]);
}
```



Tri par sélection

```
void main()
{
        int a[5]={20,30,40,50,60};
        int b[5]={60,50,40,30,20};
        int c[8]={3,8,5,4,12,13,100,12};
        unsigned i;
        tri_selection(a,5);
        for(i=0;i<5;i++)
                printf("%d\t"; a[i]);
        printf("\n");
        tri_selection(b,5);
        for(i=0;i<5;i++)
                printf("%d\t"; b[i]);
        printf("\n");
        tri_selection(c,8);
        for(i=0;i<8;i++)
                printf("%d\t"; c[i]);
}
```

La notion de table : représentation physique

A base de type struct

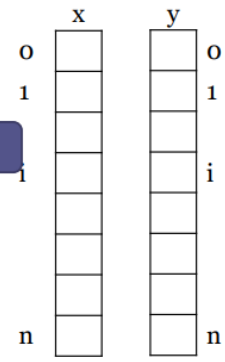
- Le type struct permet de regrouper des informations étroitement liées et éventuellement hétérogènes

```
#define n 10
struct point
{
        float x;
        float y;
};
struct point p[n];
```

<p[i].x, p[i].y> forme un point

A base de tableaux indicés en parallèle

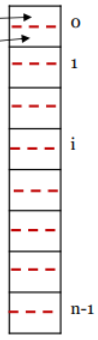
```
define n 10
float x[n];
float y[n];
```



<x[i], y[i]> forme un point

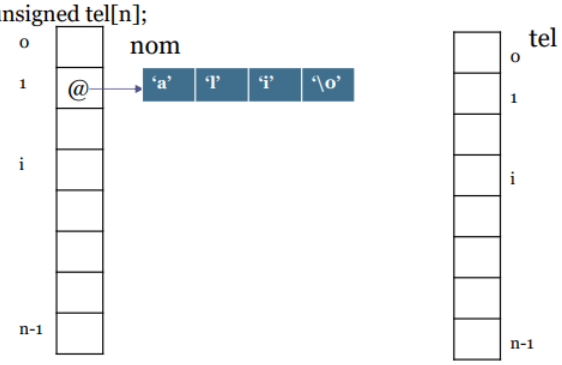
A base de struct

```
#define n 10
struct abonne
{
        char * nom;
        unsigned tel;
};
struct abonne annuaire_tel[n];
```



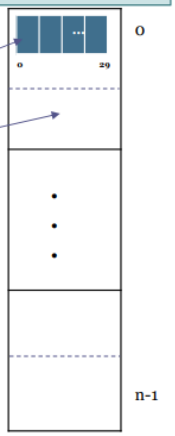
A base de tableaux indicés en parallèle

```
#define n 10
char * nom[n]; /* chaque case est un pointeur vers une chaîne de caractères */
unsigned tel[n];
```



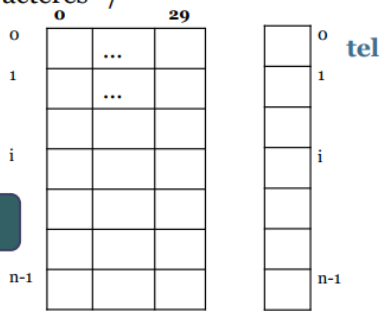
A base de struct

```
#define n 10
struct abonné
{
        char nom[30];
        unsigned tel;
};
struct abonné annuaire_tel[n];
```



A base de tableaux indicés en parallèle

```
#define n 10
char nom[n][30]; /* chaque case nom[i] est un tableau de caractères */
unsigned tel[n];
```



Matrice de caractères

Gaspillage de l'espace mémoire

Rappel des fonctions strcmp et strcpy

int strcmp (const char * chaîne1, const char * chaîne2)	
Rôle	Comparer lexicographiquement chaîne1 et chaîne 2
chaîne1	Adresse de la première chaîne
chaîne2	Adresse de la deuxième chaîne
Valeur de retour	- < 0 : si la chaîne d'adresse chaîne1 arrive avant la chaîne2 - > 0 : si la chaîne d'adresse chaîne1 arrive après la chaîne2 - = 0 : si les deux chaînes sont identiques

Recherche séquentielle : Réalisation en C

Réalisation en C : Solution 1

```
# define n 100
/* variables globales */
char * nom[n] ;
unsigned tel[n] ;
int recherche_sequentielle(char *x)
/* rend -1 si x n'appartient pas à nom sinon le numéro de tel correspondant */
{
        unsigned i ;
        /* pour parcourir la table nom */
        for (i=0 ; i<n ; i++)
        {
                if(strcmp(nom[i],x)==0)
                /* issue positive */
                return(tel[i]);
        }
        /* issue négative */
        return -1 ;
}
```

Evaluation de la Solution 1

- L'utilisateur du schéma for n'est pas approprié
  - le nombre d'itérations n'est pas connu dés le début
    - ceci explique la sortie brutale au sein du corps du schéma for

Il est recommandé d'utiliser le schéma while

Réalisation en C : Solution 2

```
#include <string.h>
#define n 10
char * nom[n];
unsigned tel[n] ;
int recherche_sequentielle(char * [])
/* rend -1 si x n'appartient pas à nom sinon le numéro de tel correspondant */
{
        unsigned i ;
        /* pour parcourir nom */
        i=0 ;
        while ((i<n) && (strcmp(x,nom[i]) !=0))
                i++;/* pour passer à l'élément suivant */
/* à la sortie de la boucle while :
( i>=n) =>échec ou strcmp(x,nom[i])==0 => succès et forcément (i<n) */
        if(i<n)
                return tel[i] ;
        else
                return -1 ;
}
```

Recherche séquentielle avec sentinelle

Evaluation de la Solution 2

- L'expression qui gouverne le schéma while est une expression composée de la forme p && q
  - l'ordre d'apparition de p et de q est significatif
    - utilisation de && dans un esprit non commutatif

```
i=0 ;
while ((i<n)&&(strcmp(x,nom[i]) !=0)
        i++;
```

- À chaque itération on fait 3 évaluations
  - 1ère évaluation : i<n →res1
  - 2ème évaluation : strcmp(x,nom[i]) !=0 →res2
  - 3ème évaluation : res1 && res2

•On souhaite simplifier l'expression composée
•Pour y parvenir, on fait appel à la technique de la sentinelle

Solution 3 : technique de la sentinelle

- Au lieu de réserver un espace de n éléments
  - on réserve un espace de N= n+1 éléments

#define N 11 /* au lieu de 10*/

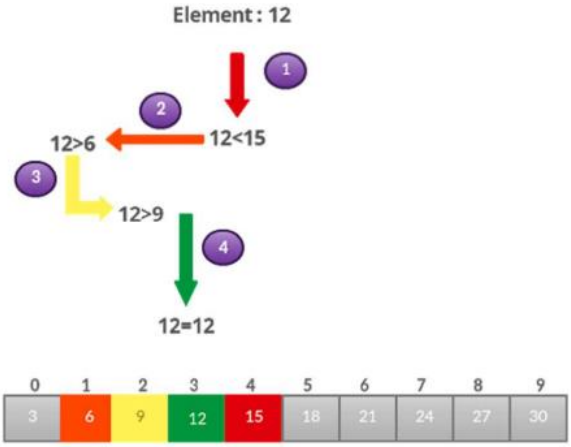
- L'élément de position N-1 joue le rôle d'une sentinelle
  - on ne peut pas aller au-delà d'une sentinelle
  - à la position N-1, on va mettre l'information « x » à rechercher dans le tableau « nom »
  - dans tel[N-1], on va mettre -1
  - « x » est appelé sentinelle

Rappel des fonctions strcpy

char * strcpy(char* but, const char * source)	
but	Adresse à laquelle sera recopiée la chaîne source
source	Adresse de la chaîne à recopier source
valeur de retour	Adresse de la chaîne but

Aucun contrôle de longueur n'est effectué par cette fonction, il est donc nécessaire que l'emplacement réservé à but soit suffisant pour y recevoir la chaîne située à l'adresse source sans oublier le caractère de fin de chaîne ('\0')

Recherche dichotomique : Exemple d'animation



Recherche dichotomique : Réalisation en C

Réalisation en C

```
#include <string.h>
#define n 100
char * nom[n];
/* nom est trié par ordre alphabétique* /
unsinged tel[n] ;
int recherche_dichotomique(char *x)
/*-1 si x n'appartient pas à nom sinon tel correspondant*/
{
        int g,d ;       /* sous tableau de recherche */
        int m ;         /* indice de l'élément au milieu */
        int comp ;/*résultat de comparaison  x et nom[m]*/

        /*initialisation*/
         g=0 ;
        d=n-1 ;
```

```
do
{
        m=(g+d)/2 ;             /* division entière */
        cmp=strcmp(x,nom[m]) ;
        if(cmp==0) ;
                return tel[m] ; /* issue positive */
    /* soit déplacer g soit déplacer d jamais déplacer les deux à
la fois */
        if(cmp<0)  /*x<nom[m]*/
                d=m-1;
        else       /*(cmp>0)=> x>nom[m]*/
                g=m+1 ;
}while(g<=d) ;
/* g>d sous tableau vide */
return -1 ;                     /* issue négative* /
}
```

Recherche dichotomique : Complexité

On note C_n : le nombre de fois où strcmp() est effectuée

$C_n = C_{n/2} + 1$
$= C_{n/2^2} + 1 + 1$
$= C_{n/2^3} + 1 + 1 + 1$
...
$= C_{n/2^i} + i$
...
$= C_1 + N = 1 + N = 1 + \log_2 n$

avec $C_1 = 1$ (pour un tableau d'un seul élément)

Ainsi, cet algorithme est en $O(\log_2 n)$

On pose $n = 2^N$ ou $N = \log_2 n$

Les actions algorithmiques simples

- L'affectation
  - Exemples

```
A ← 3
B ← A * 5
A ← B – 2
C ← A
C ← B + 6
B ← B + 1
```

Attention : il est interdit d'écrire :

A + 1 ← 3	: A+1 n'est pas un identificateur de variable
A ← B	: sans que B n'ait déjà une valeur (indéfinie)