Tableaux

Certains problèmes nécessitent beaucoup de variables du même type.

Exemple : relevé de températures matin et soir dans 10 villes pour 10 jours : 200 valeurs à stocker : déclarer 200 variables ?

Utiliser un **tableau** regroupant ces 200 valeurs : ensemble de 'cases' (de cellules mémoire) que l'on repère avec leur **numéro** ou **indice**.

Pour l'exemple précédent : une variable tableau : un groupement de 200 cases plutôt que 200 variables.

Toutes les cases ont le même type : on fera un regroupement de **char**, d'**int**, de **double**,…

un tableau est une variable : il faut la nommer.

Déclaration de tableau

Syntaxe : utiliser des crochets entre lesquels on indique le nombre d'éléments (de variables) dans le regroupement après le nom de variable.

```
Type_des_éléments   nom_du_tableau[nombre_d_éléments];
```

exemples de déclarations :

tableau nommé tab_c contenant 50 valeurs de type char :

```
char  tab_c[50];
```

tableau nommé abcd contenant 8 valeurs de type double :

```
double abcd[8];
```

tableau nommé releveTemp contenant 200 entiers longs :

```
long int releveTemp[200];
```

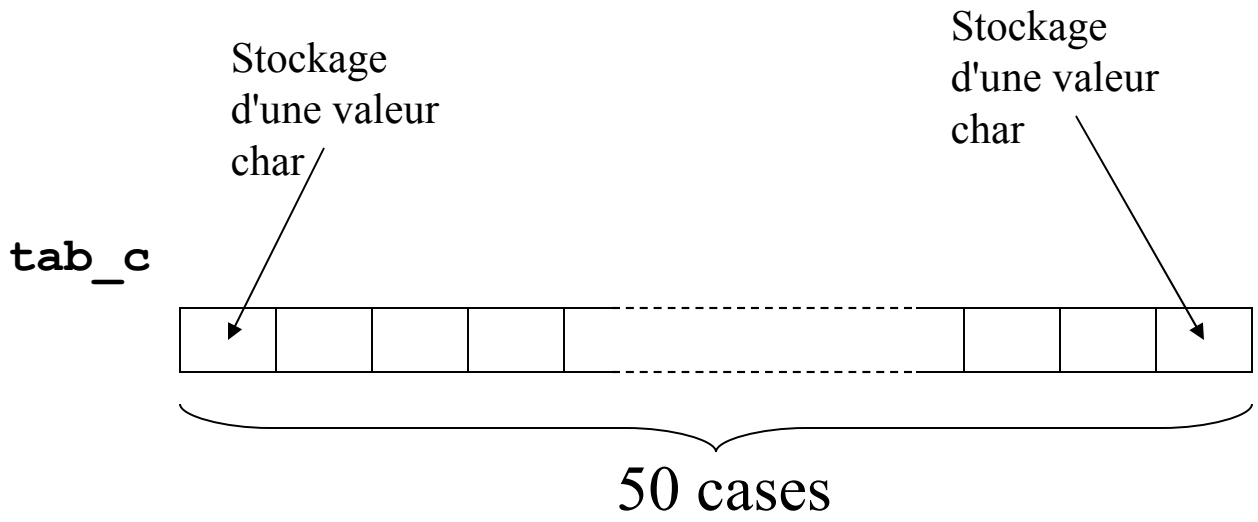
Déclaration de tableau

Le nombre d'éléments lors de la déclaration doit être une constante :

la mémoire est allouée lors de la compilation (avant l'exécution)

jamais de variable entre les crochets à la déclaration !

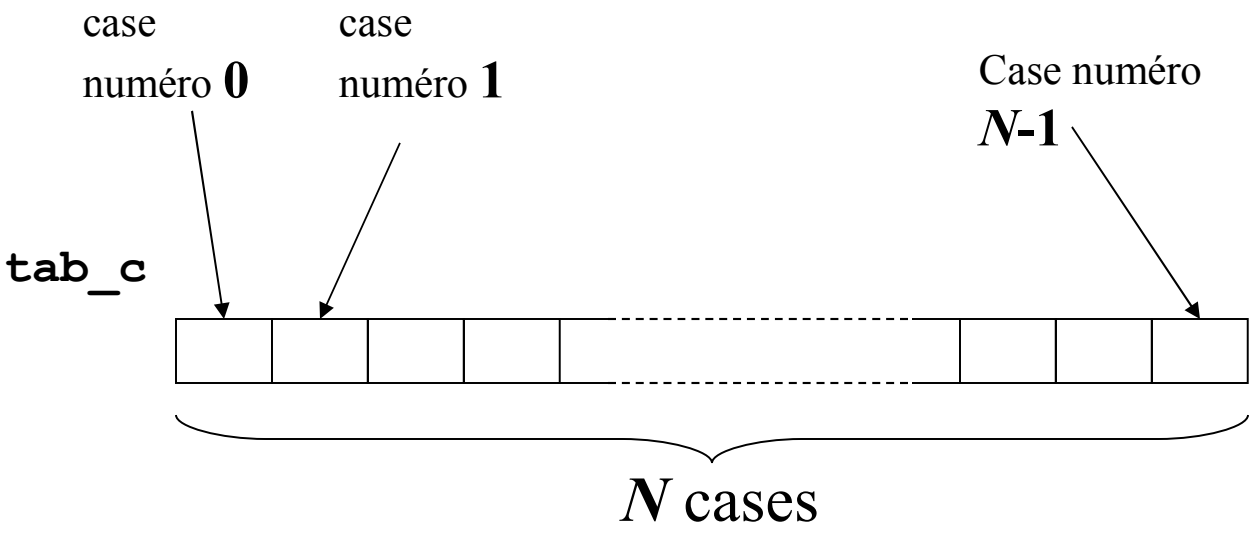
Représentation : par exemple pour la déclaration **`char tab_c[50];`**



Accès aux éléments

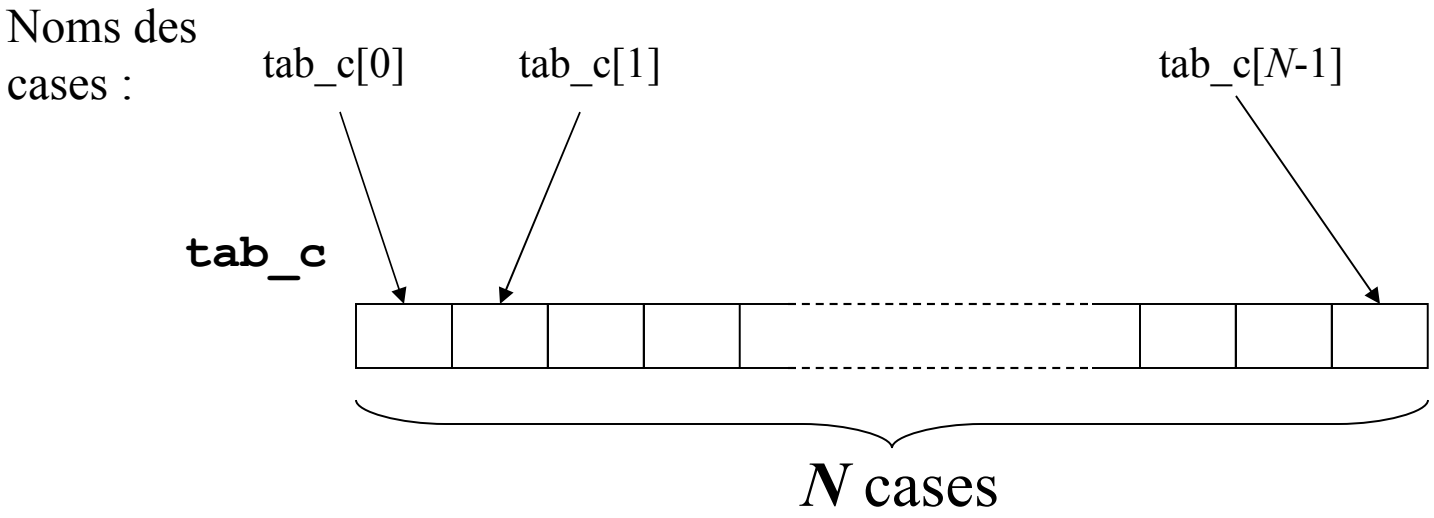
Accès aux éléments : par le numéro de la case (indice).

Soit *N* le nombre cases (*N* constant) : les indices vont de 0 à *N*-1



Accès aux éléments

Chaque case est accessible par un nom : le nom du tableau suivi de l'indice noté entre crochets. Dans une case : une variable.



Accès aux éléments

Chaque case manipulable comme une variable.

Exemple : remplissage de quelques cases d'un tableau.

```
#include <stdio.h>

void main()
{
        char tablo[10];

        tablo[0] = 'x';
        tablo[1] = 'F';
        …
        tablo[9] = 'D';
}
```

Chaque case est de type **char**

Accès aux éléments

Exemple : remplissage de quelques cases d'un tableau et affichage

```
#include <stdio.h>

void main()
{
	char tablo[10];

	tablo[0] = 'x';
	tablo[1] = 'F';
	…
	tablo[9] = 'D';

	printf("%c\n",tablo[5]);
}
```

Accès aux éléments

Dans l'exemple précédent : on utilise les éléments du tableau, pas le tableau lui-même.

Le tableau est l'ensemble de toutes les cases (ou variables), on ne peut pas lui appliquer d'opérations.

Le tableau donne accès aux éléments.

Ne jamais écrire : `tablo='xF…D';`

seulement possible pour l'initialisation.

Accès aux éléments

Ne pas confondre : tableau lui-même (c'est le regroupement ou ensemble de variables) et les éléments du tableau.

Ainsi, on n'utilise jamais, pour les manipulations de base (calcul, saisie, affichage), le tableau, mais toujours ses éléments : on doit toujours avoir la notation avec les crochets lorsqu'on veut travailler avec les valeurs dans le tableau.

S'en souvenir en cas de doute !

Nous verrons plus tard que l'on peut utiliser un tableau dans sa globalité.

Initialisation d'un tableau

Lister les valeurs des éléments lors de la déclaration (impossible après)

syntaxe :

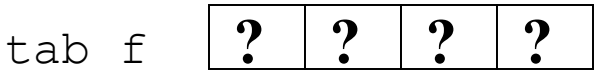
type nom_du_tableau[nb_elements]={elt_0, elt_1,…,elt_i};

effet : initialise les éléments du tableau avec les valeurs fournies. Si le nombre de valeurs fournies est inférieur au nombre d'éléments du tableau, les cases restantes sont initialisées à 0 (quel que soit le type).

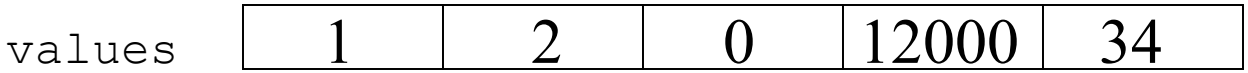
Initialisation d'un tableau

Exemples :

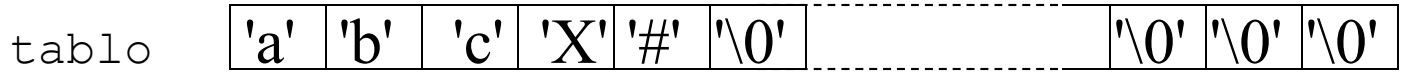
```
float                 tab_f[4];     /* cases non initialisées */
```




```
unsigned short int      values[5]={1,2,0,12000,34};
```




```
char                  tablo[20]={'a','b','c','X','#'};
```



Traitements avec les tableaux

Distinction entre taille maximale et taille utile de tableau.

Taille maximale : nombre d'éléments donné lors de la déclaration : le tableau ne pourra pas en contenir plus.

Taille utile : nombre d'éléments effectivement stockés dans le tableau : inconnu lors de la déclaration.

Pour la taille maximale : prévoir un majorant : il y aura des cases inutilisées (+ tard : comment ne pas perdre de place).

Toujours conserver le nombre de cases effectivement utilisées : taille utile du tableau : on se réfèrera tout le temps à cette dernière.

Traitements avec les tableaux

Utiliser une variable entière (toujours positive) pour la taille utile, nom significatif (taille, nbElement, nombreElem, size,…).

Illustration :

```
float tabTemp[20];
unsigned char nbElem;
/* déclaration du tableau et de sa taille utile */
```

toujours mettre à jour la taille utile en fonction des traitements effectués : **ajout / suppression / initialisation**

évite de travailler avec des valeurs inconnues.

Affichage de valeurs

Traitement de base, souvent utile : afficher tous les éléments présents dans le tableau (du début jusqu'à la taille utile, pas la taille maximum)

quelle boucle utiliser ?

Que faire dans cette boucle ?

Quelques considérations de mise en page (afficher '\n' à la suite des différentes valeurs).

Donner un exemple avec char, entier, nombres à virgule;

Traitements avec les tableaux

Exemple classique : initialiser un tableau avec des valeurs saisies au clavier. Programme de saisie de notes et de calcul de moyenne :

on saisira au maximum 10 notes, et on fera la moyenne des notes saisies.

Au maximum : renseigne sur la taille maximum du tableau

mais il peut y en avoir moins ! (on ne le saura que lors de la saisie !)

d'où l'algorithme suivant :

Traitements avec les tableaux

Déclarer le tableau de notes (10 notes au maximum); déclarer la taille utile du tableau;

on arrête la saisie des notes si l'une des deux conditions suivantes est vraie :

1) 10 notes ont été saisies (le tableau est rempli, il n'y a plus de place pour stocker les résultats)

2) il n'y a plus de notes à saisir : l'utilisateur a terminé : il faut donc demander après chaque saisie si l'utilisateur a terminé

on continuera tant que les deux conditions sont fausses

Traitements avec les tableaux

Au départ : aucune note n'est saisie, la taille utile du tableau est égale à 0; à chaque saisie, il faudra l'augmenter de 1.

Choix du type de boucle à utiliser : boucle do…while

conditions de sortie

ensuite, récapitulatif des notes et calcul de moyenne : on utilise encore une boucle : laquelle et pourquoi ? Boucle for : on connaît la taille utile.

Saisie de notes : algorithme

Taille ← 0;

faire
    | saisir une note
    | ranger la note dans le tableau
    | ajouter 1 à la taille utile
tant que (taille < 10) ou (l'utilisateur choisit de continuer)

/* affichage des notes */

pour i de 0 à taille faire
    | afficher la note
    | mettre à jour la somme des notes

calculer et afficher la moyenne

Saisie de notes : programme

```
#include <stdio.h>
void main()
{
int tabNotes[10];
int compt, taille, saisie;
int sommeNotes;
float moyenne;

taille=0;

/* saisie des notes dans le tableau */

do
        {
        printf("entrez une note entre 0 et 20 :");
        scanf("%d",&saisie);
        tabNotes[taille]=saisie;
        taille=taille+1;
        printf("voulez-vous continuer la saisie (O/N) ?");
        scanf("%c",&reponse);
        }
while ((taille < 10) && (reponse='O'))
```

Saisie de notes : programme

```
/* affichage des notes */

sommeNotes=0;

for (compt=0; compt < taille; compt++)
        {
        printf("%d ",tabNotes[compt]
        sommeNotes = sommeNotes+tabNotes[compt];
        }

/* calcul et affichage de la moyenne */

moyenne = sommeNotes/taille;

printf("la moyenne est egale a :%f\n",moyenne);
}
```

Applications : saisie directe

Programme précédent : saisie des notes dans une variable intermédiaire (saisie). Possibilité de saisie directement dans le tableau :

remplacer `scanf("%d",&saisie); tabNotes[taille]=saisie;`

par : `scanf("%d",&tabNotes[taille]);`

cependant : soucis avec le compilateur Visual : utiliser une variable intermédiaire résout le problème.

Copie de tableaux

Il est parfois utile d'avoir deux exemplaires d'un tableau, pour conserver des valeurs avant une modification par exemple : on aimerait copier un tableau (nommé tabSource par exemple) vers un autre (nommé tabDest).

Ne pas faire tabDest=tabSource ! On doit toujours travailler avec les éléments individuels : tout ce qu'on peut faire, c'est faire en sorte que les éléments de tabDest soient égaux à ceux de tabsource, et qu'ils aient la même taille utile :

algorithme de recopie de tableau :

Tableaux, recherches, tris

Algorithmes classiques sur les tableaux :

recherche d'un élément dans le tableau :

plusieurs valeurs stockées, en retrouver une : base de données

rechercher du plus petit (ou du plus grand) élément du tableau

intervalle de valeurs : graphes

trier les éléments par ordre croissant ou décroissant

classement

Tableaux, recherches, tris

Recherche d'un élément :

- cas 1 : est-il dans le tableau ? Réponse de type oui/non
- cas 2 : où est-il ? → dans quelle case ? → quel est son indice ? (entre 0 et N, N est la taille utile du tableau)

dans les deux cas : on ne sait pas si l'élément se trouve dans le tableau. Pour le cas 2 : si l'élément n'est pas dans le tableau, il n'a pas d'indice…on lui attribuera alors l'indice -1. C'est une **convention**.

Il peut y avoir plusieurs occurrences de l'élément dans le tableau : on s'arrêtera à la première.

On peut aussi compter le nombre d'occurrences dans le tableau.

Recherche d'un élément

Recherche d'un élément :

- cas 1 : est-il dans le tableau ? Réponse de type oui/non
- cas 2 : où est-il ? → dans quelle case ? → quel est son indice ? (entre 0 et N, N est la taille utile du tableau)

dans les deux cas : on ne sait pas si l'élément se trouve dans le tableau. Pour le cas 2 : si l'élément n'est pas dans le tableau, il n'a pas d'indice…on lui attribuera alors l'indice -1. C'est une **convention**.

Il peut y avoir plusieurs occurrences de l'élément dans le tableau : on s'arrêtera à la première.

On peut aussi compter le nombre d'occurrences dans le tableau.

Recherche d'un élément

Traitement du cas 1 :

3 questions à se poser :

où débuter la recherche ?

Au début du tableau : à l'indice numéro 0

Comment traiter une case pour savoir si l'élément s'y trouve ?

On compare la case du tableau avec la valeur à rechercher

Où et comment arrêter la recherche ?

Si on a trouvé la valeur : succès

Si on est en fin de tableau et que la valeur n'y est pas : échec

Recherche d'un élément

D'où l'algorithme de recherche dans un tableau :

déclarer le tableau **`Tab`**; déclarer l'indice **`ind`**; déclarer la valeur à trouver **`val`**; déclarer la taille utile **`taille`**;

```
/*saisir les valeurs de Tab et initialiser taille */
ind ← 0
saisir(val);
```

quelle boucle choisir ? On peut s'arrêter avant la fin si on trouve la valeur : on utilise une boucle **tant que** ou **faire…tant que**.

tant que ou **faire…tant que** ?

Regardons les conditions à utiliser :

Recherche d'un élément

Conditions pour la boucle : elle doit s'arrêter quand :

- la valeur est dans le tableau, dans la case que l'on est en train de traiter→ `valeur == tab[indice]`

ou quand

- on est à la fin du tableau → `indice == taille`

donc, on continue tant que : `(valeur != tab[indice])` **et** `(indice < taille)`

on utilisera donc :

```
while ((valeur != tab[indice]) && (indice < taille))
```

Recherche d'un élément

Que faire dans cette boucle ?

Elle est effectuée tant que l'on n'a pas trouvé : on doit donc passer à la case suivante, en augmentant l'indice : `indice = indice+1;` ou `indice++;`

Que faire une fois la boucle terminée ?

C'est une boucle à deux conditions : il faut vérifier quelle condition a provoqué la sortie :

si `valeur == tab[indice]` alors on a trouvé la valeur

sinon, c'est que l'on est à la fin du tableau, on n'a pas trouvé.

On utilisera donc **if…else**.

Recherche : programme

```
#include <stdio.h>

void main()
{
	short int tabVal[10]={1,-2,6,47,-12};
	char  taille=5;			/* il y a 5 éléments */
	int indice=0;
	short int valeur;

	printf("entrez la valeur a rechercher :");
	scanf("%d",&valeur);

	while((indice<taille) && (valeur != tabVal[indice]))
	{
		indice = indice+1;
	}
```

Recherche : programme

```
        if (valeur==tabVal[indice])
        {
                printf("%d est dans le tableau\n",valeur);
        }
        else
        {
                printf("%d n'est pas dans le
tableau\n",valeur);
        }
}
```

Recherche de l'indice

Modifier le programme précédent pour indiquer où se trouve l'élément (valeur) dans le tableau

Modifier le programme pour calculer le nombre d'occurrences d'une valeur dans un tableau.

Recherche du minimum/maximum

Recherche du plus petit ou du plus grand : différent de la recherche d'un élément.

Le minimum ou maximum existe forcément : pas de réponse de type oui/non.

On peut chercher :

la valeur du minimum ou maximum

son indice dans le tableau (donne de plus accès à la valeur)

Plusieurs occurrences possibles dans le tableau : pas important.

Utiliser comme variables : un tableau **`tab`**, sa taille utile **`taille`**, un **`indice`**, et la valeur du **`mini`**mum ou **`maxi`**mum recherché.

Recherche du minimum/maximum

Les 3 questions classiques :

où débuter ?

Au début du tableau : indice 0

Comment trouver le minimum/maximum ?

En comparant des éléments du tableau

Où arrêter la recherche ?

A la fin du tableau (le min/max peut être dans la dernière case utilisée)

Recherche du minimum/maximum

Parcours du tableau du début à la fin : on utilise une boucle (c'est presque toujours le cas avec les tableaux).

On connaît le nombre d'itérations : du début à la fin

boucle **tant…que** (while) ou **for** (plus pratique).

Lorsque l'on sort de la boucle : on aura trouvé le résultat : juste un affichage.

Contenu de la boucle :

approche naïve : comparer les éléments 2 à 2 pour trouver le min/max : ne donne pas le bon résultat. On trouve le min/max parmi 2 éléments, mais pas forcément celui du tableau. Comparer tab[indice] et tab[indice+1] ?

Recherche du minimum/maximum

Exemple : de indice = 0 à indice=3, comparer `tab[indice]` et `tab[indice+1]`. Le minimum sera le plus petit des deux. On fera donc un test pour savoir quel est le plus petit des deux.

Quel test ? : si (tab[indice]<tab[indice+1]) alors mini=tab[indice] sinon mini=tab[indice+1]

`Tab`	1	25	0	12	4
`indice`	0	1	2	3	
Valeurs de `mini`	1	0	0	4	

Dernière valeur calculée

Recherche du minimum/maximum

On n'a pas trouvé le minimum de tout le tableau : on n'aurait pu ne faire que la dernière comparaison. (vérifier)

Solution : garder dans mini (ou maxi) la valeur du minimum (ou maximum) le plus petit des éléments **déjà rencontrés** (et non pas les derniers).

Comparer chaque élément avec ce minimum (maximum) déjà rencontré.

Condition équivalente : si l'élément dans le tableau est plus petit que le minimum courant (plus grand que le maximum courant) : le minimum (maximum) prend la valeur de cet élément. Sinon, on ne fait rien (on garde le minimum qu'on avait avant).

Recherche du minimum/maximum

En C :

pour le minimum :

```
if (tab[indice] < mini)
        {
        mini = tab[indice]
        }
```

pour le maximum

```
if (tab[indice] > maxi)
        {
        maxi = tab[indice]
        }
```

Recherche du minimum : programme

```
#include <stdio.h>
void main()
{
	float  tab[10]={-12.3,3.14E+2,1.0E-15,0.,6.324,8.0};
	char   taille=6;
	int    indice;
	float  mini;

	for (indice=0; indice<taille;indice++)
	{
		if (tab[indice] < mini)
		{
			mini = tab[indice];
		}
	}
	printf("le minimum est : %f\n",mini);
}
```

Recherche du minimum : programme

Valeur initiale du minimum ?

1) mettre une valeur très grande : après comparaison avec une valeur du tableau, fortes chances que cette dernière soit inférieure : possibilité d'erreur.

2) initialiser avec la première valeur : pas de possibilité d'erreur.

```
        mini = tab[0];
        for (indice=1; indice<taille;indice++)
        {
                if (tab[indice] < mini)
                {
                        mini = tab[indice];
                }
        }
        printf("le minimum est : %f\n",mini);
}
```

Recherche du minimum : programme

A partir ce ce programme : recherche de l'indice du minimum dans le tableau et affichage du minimum.

A vous de jouer

Insertion non classée d'un élément

Plus simple : on doit ajouter un nouvel élément dans un tableau où se trouvent déjà d'autres éléments :

en général : insertion en fin de tableau : c'est là qu'il y a de la place ! (autres cas traités avec insertion classée).

Seule difficulté : savoir s'il reste de la place…

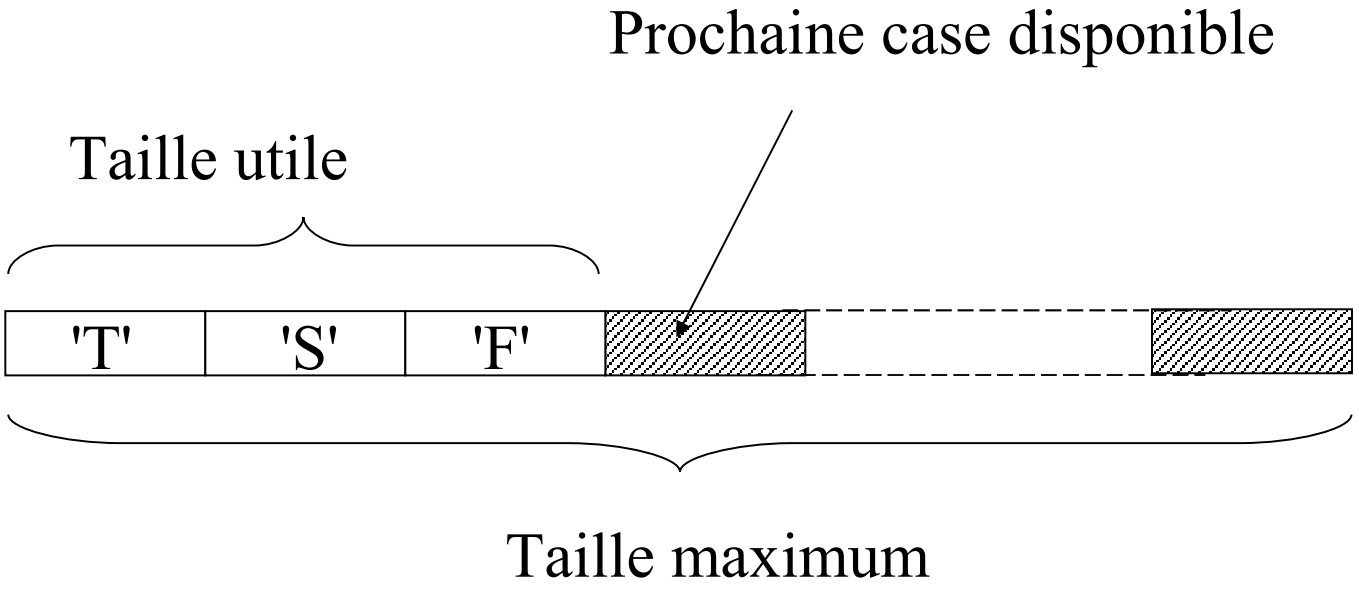
traduction de la condition précédente ?

Si taille utile <= taille maximale

garder la valeur de la taille maximale : par une variable (ou par une directive #define).

Insertion non classée d'un élément

Illustration : en gris : cases non utilisées



Insertion non classée d'un élément

Indice de la prochaine case disponible : c'est la taille utile. (indices commençant à 0)? Sur l'exemple : s'il y a 3 éléments (taille utile = 3), la quatrième case a comme indice 3 !

D'où l'insertion classée :

si taille < taille maximale alors :

tableau[taille] ← nouvel élément

taille ← taille+1

Écrire le programme correspondant avec une variable pour la taille maximale ou un #define. Il vaut mieux utiliser #define.

Insertion classée d'un élément

Méthode plus délicate : soit un tableau sont les valeurs sont déjà triées par ordre croissant ou décroissant (algo de tris vus plus tard). On dispose d'une valeur à ranger dans le tableau, et on veut que l'insertion laisse le tableau trié : trouver la place de cette valeur dans le tableau, pas forcément à la fin !

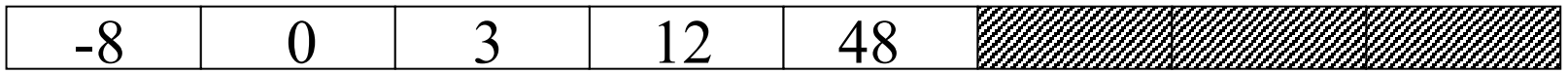
Repérer la bonne case (s'il reste une case libre au moins)

faire de la place pour le nouvel élément

le mettre à cette place.

Insertion classée d'un élément

Exemple de tableau d'entiers trié :



Taille utile = 5; taille maximum = 8

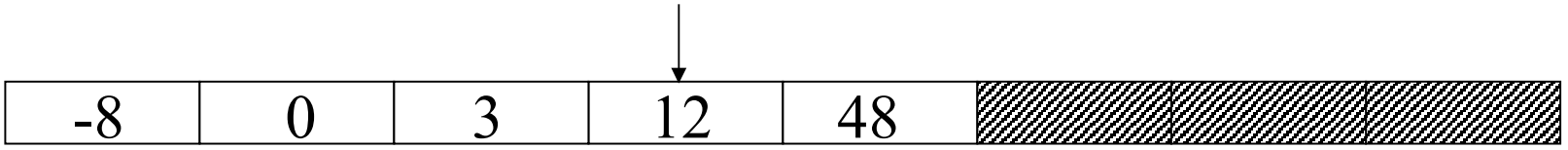
on veut insérer la valeur 9 pour obtenir comme résultat :

Le tableau est encore trié

Insertion classée d'un élément

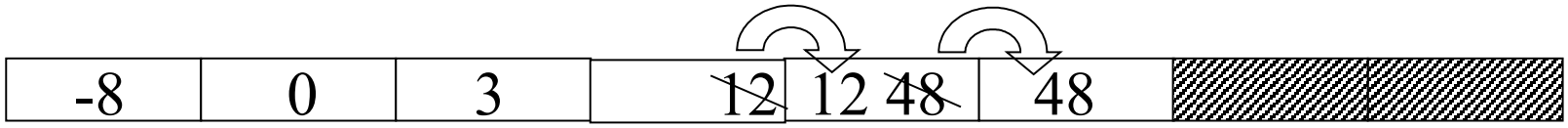
Phases intermédiaires :

repérer la case où doit être stockée la valeur à insérer



Comment trouver cette case avec un programme ou un algorithme : quelle condition ? Attention à ne pas dépasser taille utile et taille maximum !

Libérer la case voulue en décalant les éléments restants :



Insertion classée d'un élément

Phases intermédiaires :

repérer la case où doit être stockée la valeur à insérer : si le tableau est trié par ordre croissant, la valeur doit être 'à droite' (indice plus élevé) des valeurs qui lui sont inférieures et 'à gauche' (indice plus faible) que les valeurs qui lui sont supérieures.

On traite une fois de plus le tableau à partir de la première case (indice 0).

Traitement : passer à la case suivante tant que la valeur dans la case actuelle du tableau est plus petite que la valeur à insérer. (et tant qu'on ne se trouve pas à la fin du tableau).

Traduction:

```
while (tab[indice] < valeur) && (indice < taille)
        {
                indice++
        }
```

Insertion classée d'un élément

Libérer une case pour la valeur à insérer :

augmenter la taille utile (décalage des éléments).

Décaler les éléments vers la fin du tableau : attention à l'ordre du décalage !

Illustrer

choisir la bonne boucle : où débuter; où s'arrêter

donner la traduction de la boucle, puis illustrer son effet

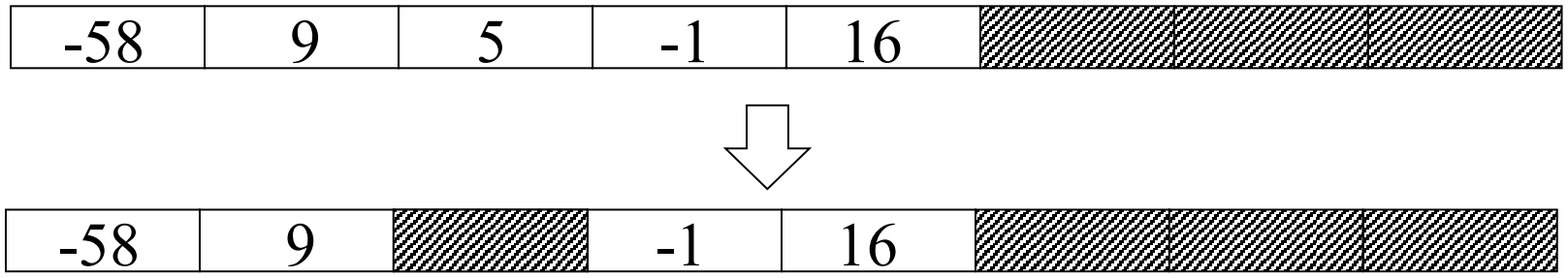
Insérer la valeur

Suppression non classée d'un élément

Très facile à faire, très difficile d'utiliser le tableau après : la suppression non classée provoque des 'trous' dans le tableau, très durs à gérer !

Comment savoir qu'une case quelconque est inutilisée ? Valeur 'spéciale', pas très élégant ni pratique !

Exemple : suppression de la valeur '5' située à la troisième case (indice=2)



Suppression non classée d'un élément

Attention à cette représentation par une case grisée : n'existe pas en C : on ne peut pas déclarer une case comme 'inutilisée'.

Que mettre à la place de '5' ?

De plus, la taille utile n'est pas modifiée, alors qu'il y a un élément de moins : très mauvaise solution.

Suppression classée d'un élément

Supprimer un élément du tableau : faire une suppression non classée puis 'recoller les morceaux'.

À l'inverse de l'insertion classée, où l'on décale les éléments pour libérer une case, on décale les éléments pour combler le 'trou' créé dans le tableau.

Il faut aussi réduire la taille utile de 1 : il y a un élément au moins.

Décalage : traduction en algo puis en C : quelle boucle, dans quel sens, où commencer, où s'arrêter.

On utilise toujours une suppression classée.

Tri de tableau

Problème classique : valeurs à ordonner de manière croissante ou décroissante (classements).

Beaucoup d'algorithmes, du plus simple au plus complexe, du plus efficace au moins efficace.

Notion de complexité algorithmique pour juger de l'efficacité en temps d'un algorithme : on donnera une indication sans entrer dans la théorie de la complexité.

Quelques idées pour trier un tableau ?

(devrait donner : tri par insertion ou extraction)

Tri à bulles

Très simple mais peu efficace : méthode "naïve"

procéder par échanges de valeurs consécutives si elles ne sont pas ordonnées 2 à 2.

Classement par ordre croissant : si deux éléments consécutifs d'un tableau ne sont pas ordonnés de manière croissante, on les échange.

Parcourir tout le tableau du début à la fin, faire ce parcours autant de fois qu'il y a d'éléments.

Algorithme : double boucle for.

Attention à la gestion des indices lorsque l'on fait des permutations, attention à utiliser une variable temporaire !

Tri à bulles

```
#include <stdio.h>
void main()
{
	char tab[6]={12,103,5,0,9,6};
	int nombreElem=6;
	int compt1, compt2;
	int echange;

	for (compt1=0; compt < nombreElem; compt1++)
	{
		for (copt2=0;compt2<nombreElem-1; compt2++)
		{
		if (tab[compt2]>tab[compt2+1])
			{
				echange=tab[compt2];
				tab[compt2]=tab[compt2+1];
				tab[compt2+1]=echange;
			}
		}
	}
}
```

Améliorations possibles ?

Si le tableau est trié, on devrait arrêter le tri : lors d'un parcours, s'il n'y a plus d'échanges de valeurs, c'est que le tableau est trié !

Compter le nombre d'échanges dans la boucle, s'il est toujours nul à la fin de la boucle, alors on peut arrêter le tri.

Vous pouvez améliorer le programme !

Tri par extraction (ou sélection)

Basé sur l'observation suivante :

lorsque le tableau est trié (par ordre croissant par exemple), on trouve le minimum à la première case, puis la deuxième valeur la plus faible dans la deuxième etc...

Principe : trouver le minimum d'un tableau et le permuter avec la valeur située dans la première case, recommencer avec le tableau sans la première case, etc...

Tri par extraction (ou sélection)

Soit un tableau comportant `t_u` variables utilisées.

On recherche la valeur minimale parmi toutes les valeurs des variables du tableau : cette valeur se trouve à un certain indice `ind_min`. Il est nécessaire de connaître la valeur du minimum ainsi que son indice, car on va devoir permuter des variables du tableau.

Il faut échanger cette valeur avec celle située dans la variable d'indice 0. Cette valeur est maintenant définitivement à sa bonne place.

Deuxième étape : on recherche l'indice de la valeur minimale parmi les valeurs des variables du tableau entre l'indice 1 (puisqu'à l'indice 0, on vient juste de ranger la bonne valeur) et l'indice `t_u-1` : on va permuter cette valeur avec celle située dans la variable d'indice 1 du tableau, et ainsi de suite...

```
#include <stdio.h>

void main()
{

	int tablo[10]={9,7,4,1,2,54}; // valeurs des variables du
tableau
	int i,j;                                                  // indices
pour les boucles
	int temp;              // variable temporaire pour échange
	int ind_min;   // indice où se situe le minimum dans le
tableau
	int t_utile;   // nombre de variables utilisées dans le
tableau

	t_utile=6;

	// affichage des valeurs avant le tri

	printf("contenu du tableau : ");
	for(i=0; i <t_utile; i++)
	{
		printf("%d . ",tablo[i]);
	}
	printf("\n");
```

```
// on passe au tri proprement dit

        for (i=0; i < t_utile; i++)
        {
                ind_min=i;      // indice du minimum à rechercher
                for (j=i+1; j<t_utile;j++)    // entre i+1 et t_utile
                {
                        if (tablo[j]<tablo[ind_min])
                        {
                                ind_min=j;
                        }
                }

                // echange
                temp=tablo[ind_min];
                tablo[ind_min]=tablo[i];
                tablo[i]=temp;
        }

        printf("contenu du tableau : ");
        for(i=0; i <t_utile; i++)
        {
                printf("%d . ",tablo[i]);
        }
        printf("\n");
}
```

Tri par insertion (du joueur de cartes)

Principe : ramener vers les première positions du tableau les valeurs en les permutant, tant qu'elles sont plus petites.

Le tri est fait en un nombre d'étapes égal au nombre de variables utilisées dans le tableau : A chaque étape i, on classe entre eux les i premiers éléments du tableau.

Exemple : soit le tableau d'entiers contenant les valeurs 8,5,7,1,2 et 9 à classer de cette manière.

Étape 1 : **8** 5 7 1 2 9

on considère que 8 est classé.

Tri par insertion (du joueur de cartes)

Étape 2 : classement des deux première valeurs : on classe ces valeurs en faisant des permutations si nécessaire :

Étape 2 : **8** ⟷ **5** 7 1 2 9

on obtient : **5** **8** 7 1 2 9

Étape 3 : on classe 7 par rapport à 5 et 8 en permutant si nécessaire

Étape 3 : **5** **8** ⟷ **7** 1 2 9

on obtient : **5** **7** **8** 1 2 9

Et ainsi de suite en classant 1 puis 2 puis 9.

Tri par insertion (du joueur de cartes)

```
#include <stdio.h>

void main()
{

        int tablo[10]={9,7,4,1,2,54};
        int i,j;                // indices pour les boucles
        int temp;               // variable temporaire pour échange
        int ind_min;
        int t_utile;

        t_utile=6;

        // affichage des valeurs avant le tri

        printf("contenu du tableau : ");
        for(i=0; i <t_utile; i++)
        {
                printf("%d . ",tablo[i]);
        }
        printf("\n");
```

Tri par insertion (du joueur de cartes)

```
        for (i=0; i < t_utile-1; i++)
        {
                for (j=i; j>=0; j--)
                {
                        if (tablo[j]>tablo[j+1])
                        {
                                temp = tablo[j];
                                tablo[j]=tablo[j+1];
                                tablo[j+1]=temp;
                        }
                }
        }

        // affichage des valeurs après le tri
        printf("contenu du tableau : ");
        for(i=0; i <t_utile; i++)
        {
                printf("%d . ",tablo[i]);
        }
        printf("\n");
}
```

Autres tris

Tri fusion, tri rapide, vus plus tard avec la récursivité : complexes à programmer malgré une définition simple, complexes à programmer et à mettre au point.

Les chaînes de caractères

Chaîne de caractères : tableau contenant une suite de caractères, utilisé pour stocker du texte en général.

On peut utiliser ces tableaux de manière un peu différente : outils spécifiques pour les manipuler, mais ne pas perdre de vue que ce sont des tableaux !

Déclaration : une **chaîne de caractères** est un tableau de caractères :

char uneChaine[25];

uneChaine est un tableau pouvant accueillir au maximum 25 caractères.

Les chaînes de caractères

Possible de les initialiser en utilisant une autre syntaxe. Par xemple, une chaîne de 10 caractères contenant le mot 'bonjour' :

char chaine[10]={'b','o','n','j','o','u','r'};

lourd à écrire !

On peut utiliser des guillemets double :

char chaine[10]="bonjour";

pour l'affichage et la saisie de chaînes de caractère, formats spéciaux avec printf et scanf : **%s** (**s** signifie **string** pour chaîne).

%s signifie que l'on va afficher l'un après l'autre tous les caractères du tableau : seule occasion où on affiche plusieurs valeurs avec un seul format !

Les chaînes de caractères

Plus pratique que d'afficher l'un après l'autre chaque caractère du tableau !

De plus, n'affiche que les caractères 'intéressants' : exemple de la chaîne contenant "bonjour".

`char chaine[10]="bonjour";` initialise les 7 premières cases avec les caractères voulus, puis les cases restantes avec la valeur 0 (déjà vu).

Or si on affiche la chaîne avec printf, seul *bonjour* sera écrit.

Astuce utilisée : le caractère valant 0 est considéré comme la fin d'une chaîne de caractère !

Ne pas confondre avec le caractère '0', dont le code est 48 !

Ce caractère se note aussi '\0'

Les chaînes de caractères

Sans utiliser de chaînes : comment afficher un message ?

Afficher caractère par caractère en utilisant une boucle :

```
#include <stdio.h>

void main()
{
char chaine[10]="bonjour";
int compteur;
compteur = 0;
while (chaine[compteur] != '\0')
       {
       printf("%c",chaine[compteur]);
       compteur = compteur+1;
       }
       printf("\n");
}
```

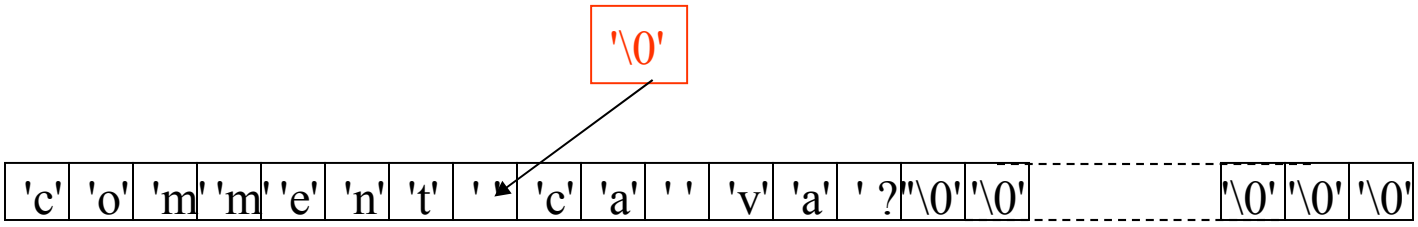
Les chaînes de caractères

```
printf("%s",chaine);
```

affiche tous les caractères présents dans le tableau chaine jusqu'au caractère '\0' (valant 0).

Si on insère le caractère '\0' dans la chaîne : exemple :

```
char chaine[25]="comment ca va?";
chaine[7] = '\0';
printf("%s",chaine);     affiche 'comment'
```



Saisie de chaîne

Le format %s est aussi utilisé avec **scanf** pour la saisie, on peut saisir directement la chaîne sans faire la saisie caractère par caractère.

Dans ce cas, on n'a pas besoin de préciser l'adresse de la chaîne, puisqu'elle représente l'ensemble des caractères. On n'utilise donc pas l'opérateur **&**.

A la saisie, un caractère '\0' est automatiquement ajouté à la fin des caractères utiles.
Exemple :

```
char chSaisie[20];
scanf("%s",chSaisie); /* ne pas ecrire : &chSaisie */
printf("%s\n",chSaisie);
```

Saisie de chaîne

Si l'utilisateur rentre le texte 'hello !', c'est la suite de caractères 'h', 'e', 'l', 'l', 'o', ' ', '!', '\0' qui sera stockée dans le tableau.

Donc c'est bien 'hello !' qui apparaît à l'écran lorsqu'on utilise **printf**.

Attention ! scanf ne vérifie pas si le nombre de caractères saisis est supérieur à la taille maximum du tableau de caractères.

Taille utile d'une chaîne : on peut récupérer le nombre de caractères utiles (affichables) d'une chaîne en utilisant **strlen**. La fonction strlen donne le nombre de caractères utile sans tenir compte du caractère de fin '\0'.

Syntaxe : strlen(chaine_de_caracteres) est une expression qui a une valeur entière.

Saisie de chaîne

Pour l'utilisation des fonctions spécifiques aux chaînes de caractère, il faut inclure un autre fichier entête que <stdio.h>, c'est le fichier <string.h>, avec la directive suivante :

#include <string.h>

Saisie de chaîne

```
#include <stdio.h>

void main()
{
    char chSaisie[20];
    int  longueur;

    scanf("%s",chSaisie); /* ne pas ecrire : &chSaisie */
    printf("%s\n",chSaisie);
    longueur = strlen(chSaisie);
    printf("la chaine %s contient %d
    caracteres\n",chSaisie, longueur);
}
```

si l'utilisateur saisit le texte 'salut', le programme affichera :

```
salut
la chaine salut contient 5 caracteres
```

Saisie de chaîne

Particularités de **scanf** : gestion assez fantasque des espaces : non pris en compte, remplacés par des retour à la ligne…
on peut utiliser une autre fonction, **gets** qui réalise une saisie de chaîne de meilleure qualité.
Pas besoin de format **%s**, **gets** est spécialement prévu pour la saisie de chaînes de caractères.

Syntaxe : gets(chaine_de_caractere);
comme scanf : ajoute automatiquement un '\0' à la fin de la chaîne.

Ne pas oublier **fflush(stdin);** avant la saisie !

Manipulations : transferts

Copier une chaîne dans une autre : on ne peut pas affecter directement un tableau à un autre tableau, il faut donc recopier les éléments individuellement d'un tableau à un autre, ou utiliser des fonctions spécifiques aux chaînes de caractères.

Il existe une fonction recopiant les caractères utiles ainsi que le caractère de fin '\0' d'une chaîne de caractères dans une autre : strcpy (**str**ing **c**o**py**).

Syntaxe : strcpy(chaine_destination, chaine_source);

attention au sens du transfert !
strcpy ne vérifie pas si la chaîne de destination est assez longue pour accueillir les caractères.

Manipulations : transferts

Exemples :

```
#include <stdio.h>
#include <string.h>

void main()
{
	char chSource[15];
	char chDest[15];

	fflush(stdin);
	gets(chSource);
	strcpy(chDest,chSource);

	printf("ce qui a ete recopie :%s\n",chDest);
}
```

Opérateur de comparaison

Peut-on savoir si deux chaînes sont égales ? Pas intéressant, deux tableaux ne sont pas égaux, on regarde seulement si leurs éléments sont égaux. Là encore, comparaison élément par élément ou utilisation d'une fonction spécifique, nommée **strcmp** (**str**ing **c**o**mp**are).

Syntaxe : strcmp(chaine_1, chaine2); est une expression de type entier qui vaut :

- 0 si les deux chaînes ont le même contenu (semblables jusqu'au caractère '\0');
- <0 si chaine_1 est inférieure à chaine_2
- >0 si chaine_2 est inférieure à chaine_1

inférieur dans le sens lexicographique (classement du dictionnaire)

Opérateur de comparaison

Exemple :

```
#include <stdio.h>
#include <string.h>

void main()
{
	char stringOne[15]="fromage";
	char stringTwo[15]="dessert";

	/* attention dans l'ordre lexicographique les */
	/* majuscules se trouvent avant les minuscules */

	int compar;

	compar=strcmp(stringOne,stringTwo);
```

Opérateur de comparaison

Exemple (continué) :

```
        if (compar == 0)
        {
                printf("les deux chaines sont identiques\n");
        }
        else if(compar <0)  /* donc compar != 0 */
        {
                printf("%s vient avant %s dans le
dico\n",stringOne,stringTwo);
        }
        else   /* donc forcement : compar >0 */
        {
                printf("%s vient avant %s dans le
dico\n",stringTwo,stringOne);

        }
}
```

Opérateur de concaténation

Opération parfois utile : mettre bout à bout 2 chaînes de caractères (concaténer) pour faire par exemple des messages d'accueil, ou pour construire des noms de fichiers.

Tentation : utiliser + pour concaténer les chaînes : pas autorisé.

On utilise la fonction **strcat**.

Syntaxe : strcat(chaine_dest, chaine_source);

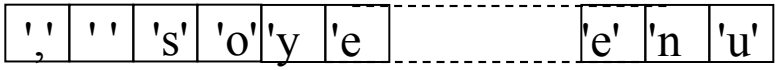
effet : ajoute les éléments de la chaîne chaine_source **après** les éléments de la chaîne chaine_dest (à partir du caractère '\0' de chaine_dest, ce caractère est écrasé).

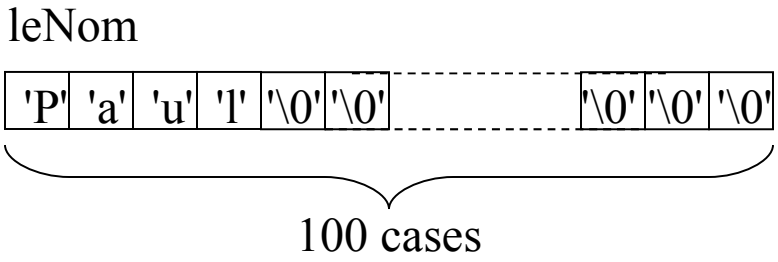
Chaine_dest est donc modifiée par l'emploi de strcat.

Opérateur de concaténation

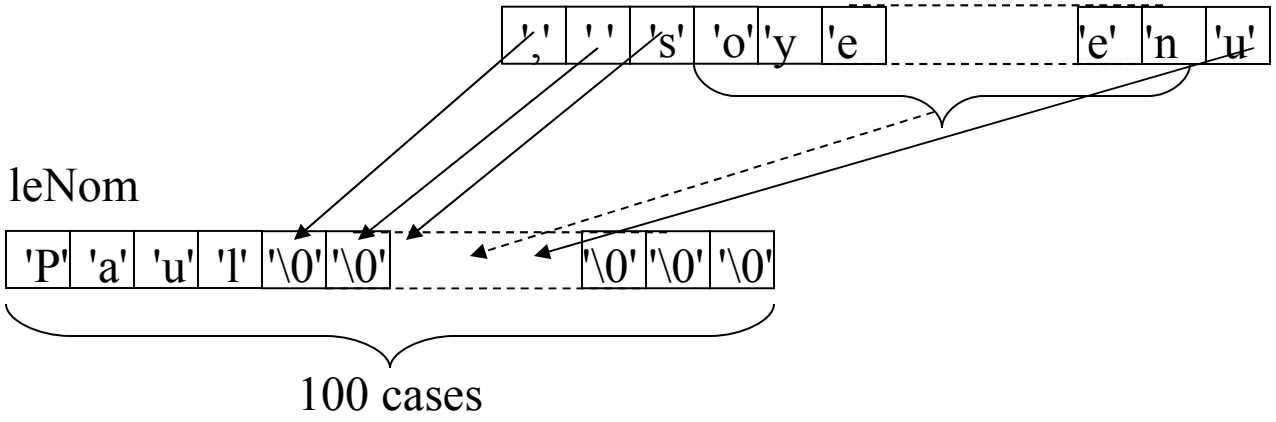
Exemple :

```
#include <stdio.h>
#include <string.h>

void main()
{
        char leNom[100];

        printf("entrez votre nom :");
        gets(leNom); /* exemple: "Paul" */
        strcat(leNom,", soyez le bienvenu");
        printf("%s",leNom);
}
```





Opérateur de concaténation



Autres manipulations

Copie d'un nombre donné de caractères :**strncpy**

strncpy(chaine_dest, chaine_source, nb_carac); :comme strcpy, mais concerne les nb_carac premiers caractères de la chaîne

comparaison d'un certain nombre de caractères : **strncmp**

strncmp(chaine1, chaine2, nb_carac); : comme strcmp, mais concerne seulement les nb_carac premiers caractères de la chaîne.

Applications :

Vérification d'un mot de passe : saisir 2 chaînes de caractères et vérifier qu'elles ont le même contenu.

Message d'accueil avec le nom : "bonjour, *untel*, comment ca va" ?

Le javanais : remplacer toutes les voyelles d'une chaîne par une seule voyelle (traitement caractère par caractère).

Le jeu du pendu : saisir un mot à deviner.

Conversion caractères/numérique

Besoin parfois de convertir une chaîne de caractères qui contient des chiffres et un point décimal en une valeur entière ou à virgule, ou de faire la conversion valeur entière ou à virgule en chaîne de caractères.

La chaîne de caractères "123" n'a pas la valeur entière 123 !

fonctions **atoi**, **atol**, **atof**

atoi : convertit une chaîne de caractère en une valeur int
valeur entière = atoi(chaine);

atol : convertit une chaîne de caractères en une valeur long int
valeur entier long = atol(chaine);

atof : convertit une chaîne de caractères en une valeur double
valeur double = atof(chaine);

Conversion caractères/numérique

La chaine de caractères doit commencer par un ensemble de chiffres (avec le point décimal éventuellement pour atof). La conversion s'arrête lorsqu'on arrive en fin de chaîne où lorsqu'un caractère non numérique est rencontré.

Exemples :

atoi("123AAAA"); donne 123

atol("15X432"); donne 15

atof("1.732+1.2"); donne 1.732

ces fonctions ne convertissent pas des expressions !

Conversion caractères/numérique

Conversion de valeurs entières ou à virgule en chaîne de caractères :

on utilise sprintf, qui est une extension de printf. Plutôt que de faire l'affichage à l'écran, on va faire cet 'affichage' dans une chaîne de caractères.

Retour sur la notion d'affichage :

à l'écran, on ne peut écrire que des caractères ! C'est le format choisi dans le printf qui va permettre une bonne mise en forme de la valeur. Ainsi le format %f permet de choisir les caractères correspondants à une valeur entière.

Exemple : printf("%",123); va permettre d'afficher à l'écran les caractère '1' puis '2' puis '3'. La conversion est donc faite par printf.

Conversion caractères/numérique

Faire cette conversion vers une chaîne de caractères avec sprintf :

sprintf(chaine_dest,chaine_de_format,valeurs);

il suffit d'ajouter la chaîne de destination, le résultat y sera stocké, il n'y a pas d'affichage à l'écran.