

TP1:

Pointeurs & Allocation de mémoire

Exercice 1

Ecrire un programme avec une fonction **void saisie(int * tab, int * N)** qui permet de saisir un tableau **tab** avec une taille **N** (entre 1 et 50), ainsi qu'une autre fonction **void affiche(int * tab, int * N)** qui affiche son contenu.

Dans la fonction main() appeler la première fonction pour saisir un tableau puis la deuxième pour l'afficher.

Exercice 2

En se basant sur les deux fonctions de l'Exercice 1, écrire un programme qui demande la saisie d'un tableau d'entiers -après avoir saisi sa taille. On affiche le tableau, puis l'utilisateur va saisir une valeur à rechercher via une fonction de recherche. Cette fonction renvoie en arguments: la position de la dernière occurrence de la valeur recherchée, ainsi que le nombre d'occurrences. Si la valeur saisie n'existe pas, un message sera affiché.

Prototype de la fonction:

void chercherVal (int * tab, int N, int val, int *pos, int *nb_occ);

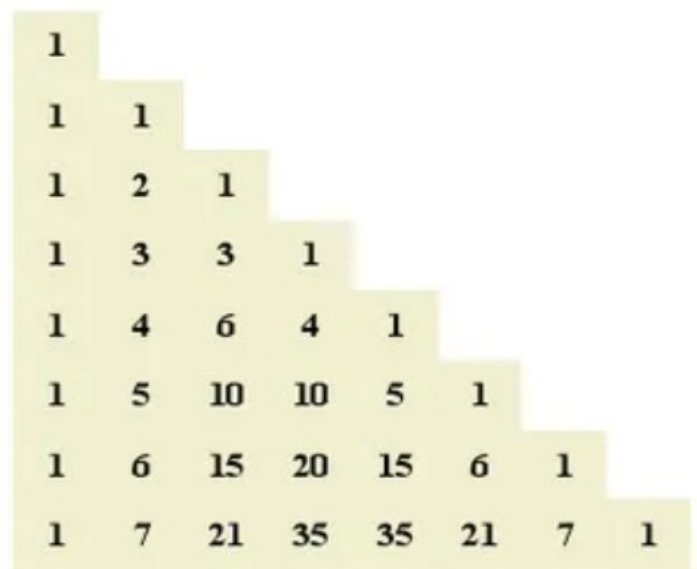
Exemple, pour le cas suivant:

11	4	58	11	99
----	---	----	----	----

La fonction donnera pour la valeur 11: dernière occurrence: indice 3, et le nombre d'occurrences: 2

Exercice 3

Le triangle de Pascal est un ensemble triangulaire comme indiquer dans la Figure:



A Pascal's triangle displayed as a grid of numbers. The triangle is right-aligned, with the first row containing a single '1'. Each subsequent row starts with a '1' and ends with a '1'. The numbers in between are the sum of the two numbers directly above them. The rows are highlighted in a light yellow color.

1							
1	1						
1	2	1					
1	3	3	1				
1	4	6	4	1			
1	5	10	10	5	1		
1	6	15	20	15	6	1	
1	7	21	35	35	21	7	1

Après avoir extrait la relation entre ses éléments, écrire un programme qui arrive à l'afficher en donnant le rang auquel on s'arrête (dans la figure, on s'est arrêté au rang 8). Pour optimiser la mémoire, on procède à l'allocation dynamique, pour cela:

- Une première fonction va allouer dynamiquement de la mémoire pour une matrice triangulaire carrée de taille donnée:

int ** alloc_mat_triang(int rang)

- Une deuxième fonction va remplir cette matrice par la relation extraite de la figure:

int ** remplir_mat_pascal(int rang)

- Une troisième fonction va afficher le résultat.

void affich(int ** mat, int rang)

- Une quatrième fonction va libérer la mémoire allouée:

void free_mat(int ** mat, int rang)