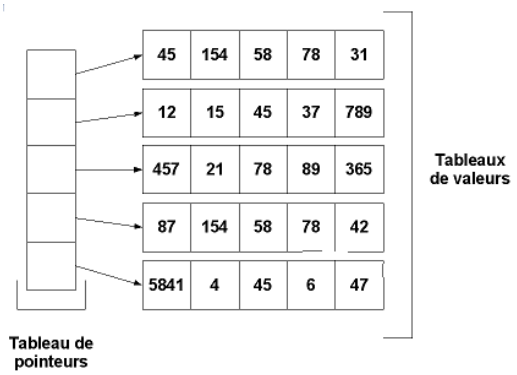
Mini-projet
Les étudiants devront présenter une démonstration de leur travail sous forme de soutenance

Dans ce mini-projet on vous propose d'implémenter le jeu du démineur. Il s'agit pour le joueur d'incarner un agent de l'équipe de déminage chargé de sécuriser un champ de mines. Le champs de mines peut être représenté sous la forme d'un tableau à deux dimensions (une matrice). En C++ on peut modéliser ceci comme suis :



Un tableau d'entiers est de type ***« int *»*** en C++ : pour allouer un tableau de m entiers :

int *p=new int[m] ;

Un tableau de pointeurs sur entiers est de type **« int * *»** : pour allouer un tableau de n pointeurs sur entiers :

int * *p=new **int *** [n] ;

Voici la fonction qui permet de créer la structure de la figure ci-dessus :

```
int * *creer_tab2D(int n, int m)
{
        int * *p=new int *[n] ;
        for(int i=0;i<n;i++) p[i]=new int[m] ;
        return (p);
}
```

Cette méthode de construction est intéressante car elle offre une simplicité d'accès aux élément de la matrice :

```
int * *T=creer_tab2D(50, 100) ;
T[5][8] ; // permet d'accéder à la ligne 5 et à la colonne 8.
```

Voici une fonction d'initialisation par des zéros :

Le jeu peut être définie par une structure matrice est une structure :

```
struct matrice
{
        int nbl ;
        int nbc ;
        int * *T ;
} ;
```

Voici une fonction qui permet de créer une matrice :

```
matrice *Creer_matrice(int n , int m)
{
        matrice *p=new matrice ;
        (*p).nbl=n ;
        (*p).nbc=m ;
        (*p).T=Creer_tab2D(n,m) ;
        return (p) ;
}
```

On peut aussi initialiser la matrice grâce à la fonction suivante :

```
void initZero_mat(matrice & J)
{
        for(int i=0;i<J.nbl;i++)
                for(int j=0;j<J.nbc;j++) J.T[i][j]=0 ;
}
```

Voici un main (à titre d'exemple) :

```
int main()
{
        matrice *J=Creer_matrice(20, 30) ;
        initZero_mat((*J)) ;
}
```

////////// Questions de révision///////

Ecrire une fonction qui prend en argument un pointeur sur une matrice et une valeur entière et qui renvoie vrai si la valeur appartient à la matrice.

Ecrire une fonction qui prend en argument un pointeur sur une matrice et une valeur et qui renvoie les indices (i,j) iù se trouve la valeur. Si la valeur n'appartient pas au à la matrice elle renvoie (-1,-1).

Testez

//

Dans le jeu, deux actions sont possibles, creuser ou poser un drapeau. Lorsque vous creusez sur une case du tableau, il n'y a que deux possibilités :

- **La case contient une mine**, qui explose et la partie est terminée
- **La case ne contient pas de mine**, et elle vous indique le nombre de cases adjacentes qui contiennent une mine.

La seconde action possible dans le jeu, **poser un drapeau**, elle constitue une sécurité pour le joueur. Lorsque l'on pose un drapeau sur une case, celle-ci affiche un symbole "attention", et on ne peut plus creuser dessus tant que le drapeau s'y trouve.

Nous souhaitons modéliser le jeu en utilisant une matrices contenant une combinaison d'informations. Une case (i,j) contient :

0 : s'il n'y pas de mine et la case n'est pas encore creusée
1 : s'il n'y pas de mine et la case a été creusée
2 : s'il y a une mine
3 : s'il n'y pas de mine et un drapeau a été posé
4 : s'il y a une mine et un drapeau a été posé

1- Ecrire un fonction « ***Int_jeu*** » qui renvoie un terrain de mines d'une taille donnée (n,m). Les mines doivent être disposées aléatoirement dans le terrain (avec une proportion donnée par le paramètre difficulté) : une case contiendra la valeur 2 si elle contient une mine et la valeur 0 sinon (au début aucune case n'est creusée et aucun drapeau n'est posé).

2- Écrire une fonction « ***mines_autour*** » qui renvoie une nouvelle matrice M de même taille que T contenant pour chaque case (i,j) le nombre de mines dans le voisinage immédiat. Ceci nous permettra d'avoir cette information tout le long du jeu et ainsi éviter le calcul à chaque étape du jeu ou à chaque affichage.

3- Écrire une fonction d'affichage permettant de présenter le plateau de jeu au joueur : chaque case du plateau contiendra une valeur indiquant le nombre de mines autour, un drapeau, ou un symbole indiquant qu'elle est encore cachée. Pour faciliter le repérage des numéros de lignes et de colonnes (par le joueur), il serait judicieux de les afficher aussi. En plus du terrain, cette fonction doit prendre comme argument la matrice renvoyé par la fonction ***mines_autour(..).***

```
   0  1  2  3  4  5  6  7  7
0  ?  ?  1….
1  ?  ?  ?…..d
2   .  .  .
```

4- Écrire la fonction ***poser_drapeau*** et la fonction ***lever_drapeau***

5- Écrire la fonction ***creuser*** qui prend en argument ***:***

- (i,j) les coordonnées de la case à creuser,
- N le nombre de cases déjà découvertes,
- T le terrain de jeu et M la matrice des nombres de mines autour des cases.

Elle renvoie true si la case ne contient pas une mine et false sinon. Elle renvoie aussi le nombre de mines découvertes. Si ce dernier atteint le nombre de cases libres alors le jeu se termine avec succès. Une case contenant un drapeau ne peut pas être creusée. Bien sûr cette fonction doit faire toutes les vérifications nécessaires avant de creuser, par exemple on ne creuse pas une case déjà creusée, on ne creuse pas une case contenant un drapeau, on ne creuse pas une case inexistante, etc.

6- Extension de creuser : si une case creusée ne contient aucune mine dans son entourage immédiat, alors vous devez creuser automatiquement partout autour d'elle. Ce processus doit se répéter pour toutes les cases nouvellement creusées et ne contenant pas de mines autour d'elles.

Écrire une fonction récursive ***creuser_recu***, permettant de faire ce travail.

7- Établir une stratégie pour donner un score à la fin d'une partie qui combinent par exemple la proportion de cases contenant des mines, la proportion des cases découvertes parmi celles ne contenant pas de mines et le temps passé à jouer. Vous pouvez aussi ajouter d'autres critères (soyez imaginatifs). Pour rendre le jeu plus intéressant, on pourrait introduire un système de jockers, l'utilisation par le joueur d'un jocker permet de découvrir une case sans risque d'explosion. Le nombre de jockers doit, à ce moment là, être utilisé pour donner un score.

8- Ecrire le programme correspondant au jeu démineur en utilisant les fonctions demandées précédemment.

10- Facultatif : dans la version de base de ce jeu, le joueur meurt dès qu'il tente de creuser une case contenant une mine. Nous vous proposons d'écrire une version dans laquelle les mines ont une puissance et l'utilisateur est muni d'un indicateur de santé. Quand le joueur tente de creuser une case contenant une mine avec une puissance donnée, sa santé est réduite d'une valeur proportionnelle à la puissance de la mine. La partie se termine quand le joueur atteint une santé nulle. Ici encore il faut donner une formule pour le calcul du score.