Université Paul Sabatier

Licence Sciences Appliquées

EEA / ISS / Mécanique / Génie Civil

Deuxième Année

Travaux Dirigés d'Informatique

P. Castelan / D. Mary / D. Buso - version 1.1 - 2019

MISES EN GARDE IMPORTANTES

Vous allez utiliser une salle équipée d'ordinateurs, l'Université met à votre disposition du matériel qui est coûteux. Cela implique que vous respectiez le matériel et ses utilisateurs.

Il est **formellement interdit** de modifier les réglages écran, de naviguer sur le web durant les heures de TP, d'installer un logiciel et/ou de modifier de quelque façon que ce soit la configuration des machines. *Tout étudiant pris sur le fait sera exclu de la séance de TP.*

De plus, pour des raisons de courtoisie, il vous est demandé d'éteindre vos téléphones portables en entrant en salle de TP.

Vous ne pouvez vous contenter de travailler que durant les TP. Vous devez travailler entre les séances. Le langage C est une langue que vous devez apprendre. On n'apprend pas une langue en ne travaillant que 2h par semaine !

Cet enseignement est compliqué : il vous faut apprendre le langage, mais aussi un nouveau mode de raisonnement pour traiter les problèmes posés.

L'équipe met à votre disposition un site web : http://eeacastelan.ups.free.fr/. Ce semestre, vous utiliserez les deux liens suivants :

- **L2 BASES C :** Ce blog contient des informations sur l'enseignement, les textes des TP, en fin de semestre les corrections des TP.
- **Fiches Langage C :** Ce blog contient des fiches résumant les principales instructions de langage C que vous devez connaître. Certaines fiches sont indiquées (S4) cela signifie que ces fiches font référence à des parties du cours d'informatique que vous verrez le semestre prochain.

Si vous voulez tirer profit de ces TD, il est impératif que vous les prépariez avant de venir en séance.

Cet enseignement fonctionne avec un "contrat de confiance". Il s'agit d'une collection d'exercices qui vous seront donnés durant le semestre et qui ressemblent aux sujets d'examen.

Vous êtes venus à l'Université pour apprendre, ne laissez pas passer votre chance. Si vous rencontrez des difficultés, n'hésitez pas à questionner votre enseignant. En dernier recours vous pouvez contacter M. Mary ou M. Castelan (par email uniquement).

Nous vous rappelons également que les **TPs sont obligatoires** et que toute absence doit être justifiée, auprès du responsable Mr Buso, et rattrapée. Les étudiants qui auraient plus d'une absence injustifiée au cours du semestre ne seront pas convoqué au Contrôle Terminal de TP.

Les rattrapages peuvent se faire avec l'accord de l'enseignant en séance sous réserve de places disponibles. Attention, tout changement de groupe est interdit sous peine de non convocation à l'examen.

TD 1 - L'environnement de développement.

Le système avec lequel vous travaillerez est un système UNIX dérivé d'une version de LINUX (Darwin pour être complet). Vous utiliserez le compilateur gcc et un éditeur de texte spécialisé dans l'édition de programmes : textwrangler.

Ce sont ces éléments que nous allons vous présenter lors de ce premier TD. Vous trouverez sur le site pédagogique des liens pour aller chercher le compilateur et l'éditeur qui fonctionnent sous windows (gcc et notepad++).

L'éditeur de texte

Il s'agit d'un programme en licence libre pour les universités et particuliers nommé TextWrangler. Son icône est un losange bleu.

Un simple clic sur l'icône dans le Dock (la barre à droite de l'écran) permet le lancement de cet application.

La première action va consister à écrire un programme. Saisissez le code suivant dans la fenêtre :

```
#include<stdio.h>
int main(void)
{  printf("Coucou \n");
   return 0;
}
```

Pour obtenir les caractères "spéciaux" les combinaisons de touches sont les suivantes :

Caractère	Touche	Caractère	Touche
{	alt⌥ + (	]	alt⌥ +⇧+)
}	alt⌥ +)	\	alt⌥ +⇧+ /
[	alt⌥ +⇧+ (	| (pipe)	alt⌥ +⇧+ L

Une fois le code saisi, enregistrez-le soit en déroulant le menu "Fichier" et en choisissant l'item "Enregistrer", soit en tapant au clavier le raccourci "⌘" + S.

Une fenêtre se déroule au dessus du document, naviguez jusqu'au dossier "Bureau".

Enregistrez sous le nom qui vous convient en n'oubliant pas de faire suivre le nom du suffixe ".c" ('c' **minuscule** et non 'C' majuscule).

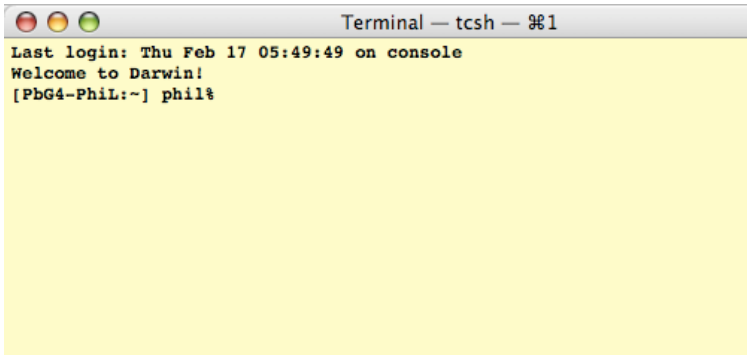
IMPORTANT : Veillez à ne pas utilisez pas de caractère espace dans le nom du fichier, ni de caractère accentués ou de signe de ponctuation, parenthèse, etc. Vous n'avez le droit qu'aux 26 lettres de l'alphabet, au caractère souligné et aux chiffres.

Le Terminal

Une fois le programme sauvegardé, il vous faudra le compiler pour pouvoir exécuter le programme que vous venez d'écrire. La compilation se fait dans une autre fenêtre appelée "Terminal".

Pour lancer le terminal, cliquez sur son icône dans le Dock (un moniteur noir avec un prompt blanc “>”)

Vous devez obtenir une fenêtre qui doit ressembler à ceci :



Cette fenêtre est un accès à la ligne de commande UNIX, un peu comme la fenêtre MS_-DOS dans un environnement Windows.

La première chose à faire une fois le terminal lancé, est d’aller dans le dossier de travail qui est le Bureau : pour ce faire tapez :

```
cd Desktop
```

Puis vérifiez que vous vous trouvez bien dans le dossier travail : tapez

```
pwd
```

cette commande (Print Working Directory) affiche le chemin d’accès au dossier où vous vous trouvez. Cela doit ressembler peu ou prou à :

```
/Etudiant_1/etudiant/Desktop
```

Notez que contrairement à Windows, le séparateur sous UNIX est “/” et non “\”.

Les commandes suivantes peuvent vous être utiles pour travailler sous UNIX :

Commande	Action
pwd	affiche le dossier courant
cd	change de dossier, taper “ ..” pour revenir au dossier parent
ls	affiche le contenu du dossier courant
clear	efface la fenêtre du terminal (il est possible avec l’ascenseur sur le coté de voir les résultats précédents, par défaut 10 000 lignes sont gardées)
↑	fait défiler les commandes saisies précédemment (remonte dans la liste)
↓	fait défiler les commandes saisies (descend dans la liste)

Le compilateur

Pour compiler un programme il suffit de taper la commande **cc** suivie du nom du fichier.

La version du Shell de l'UNIX que vous utilisez comporte une fonction d'auto-complétion. Cela veut dire que le système est capable s'il n'y a pas ambiguïté de compléter tout seul la commande que vous tapez.

Par exemple, imaginons que vous ayez sauvegardé le programme saisi au point 1/ sous le nom toto.c. Tapez alors

```
cc t
```

puis appuyez sur la touche tabulation ("⇥")

Le nom *toto.c* apparaît alors, vous n'avez plus qu'à valider en appuyant sur la touche "↩".

Le programme est alors compilé et un fichier nommé **a.out** est créé. Ce fichier est le programme. Vérifiez que ce fichier existe bien à l'aide de la commande **ls** (vous devez aussi le voir sur le bureau)

Pour vérifier que le programme fonctionne bien, il suffit de le lancer à l'aide de la commande :

```
./a.out
```

Vous devriez obtenir ceci :

```
Coucou
```

Bien entendu, il est possible que vous ayez commis des erreurs en saisissant le programme. Par exemple, omettez le point virgule après la commande printf dans le programme toto.c. Le code doit être celui-ci.

```
#include<stdio.h>
int main(void)
{  printf("Coucou \n")
   return 0;
}
```

➡ Cliquez dans la fenêtre de TextWrangler, et modifiez le code. Puis enregistrez-le.

Notez que si le document n'est pas enregistré, il apparaît un point noir dans le bouton rouge en haut à gauche de la fenêtre de l'éditeur.

➡ Activez le terminal (cliquez sur sa fenêtre ou sur son icône dans le Dock), puis compilez à nouveau le programme.

```
cc toto.c
```

Vous devriez obtenir quelque-chose ressemblant à ceci (suivant la version du compilateur installée) :

```
toto.c: In function 'main':
toto.c:5: error: parse error before 'return'
```

Note Bene:

*Il est possible d'arrêter un programme à tout instant en tapant au clavier "**ctrl + C**". Il est aussi possible de fermer la fenêtre du terminal, il suffit, alors, d'en ouvrir une nouvelle et de ne pas oublier de rendre le bureau dossier courant. (cd Desktop)*

Apprentissages de Base

Exercice 1

Objectif : Ecrire un programme simple qui mette en œuvre les fonctions **printf** et **scanf**. utilisation de l'instruction if pour des tests simples, et de **switch**. Étude des sorties formatées avec **printf**.

- Ecrivez un programme qui demande à l'utilisateur de saisir son âge, puis affiche le message, "vous avez [la valeur saisie] ans".
- Ecrivez un programme qui demande à l'utilisateur de saisir deux valeurs entières relatives puis affiche leur somme.
- Ecrivez un programme qui demande à l'utilisateur de saisir deux valeurs entières relatives A et B puis affiche leur différence.
  Complétez ce programme pour qu'il affiche l'un des trois messages suivant le cas :

```
"La différence A - B est plus grande que A"
"La différence A - B est plus petite que A"
"La différence A - B est nulle".
```

Dans les messages ci-dessus, A et B seront remplacées par les valeurs saisies.

- Complétez le programme précédent pour qu'il saisisse des valeurs réelles. L'affichage des valeurs dans les trois messages se fera alors sur 8 caractères et 3 décimales.
- Ecrivez un programme qui saisisse deux valeurs puis demande l'opération à effectuer : les réponses possibles seront +, -, * ou /. Vous utiliserez d'abord une série de test puis un switch pour choisir le type de calcul à effectuer.
- A ce point vous devez être capable d'utiliser les instructions printf et scanf sans réfléchir à leur syntaxe. Vous devez connaître la syntaxe de l'instruction switch

Exercice 2

Objectif : Ecrire un programme qui permette le calcul d'une somme d'entiers saisis au clavier. découverte des test logiques et des boucles **for** et **while/do while**. Règles d'usage de ces différentes boucles.

On se propose d'écrire un programme qui calcule la somme de N entiers entre 0 et 20 La saisie des nombres et le calcul de la somme se feront dans une même boucle.

- Dans une première version du programme, N est demandé à l'utilisateur avant la saisie des valeurs
- Dans une seconde version du programme, la saisie des valeurs dure tant que l'on n'entre pas un code d'arrêt, ce code sera le nombre 99
- Prévoyez un test pour que x, la valeur saisie, reste dans les limites imposées $0 \leq x \leq 20$, d'abord avec le programme du point 1, puis avec le programme du point 2.

Exercice 3

Objectifs : Ecrire un programme qui calcule une moyenne, comprendre l'adaptation de l'opérande à l'opérateur, apprendre à réutiliser un programme précédent.

- Editez le programme du point 2 de l'exercice 2. Modifiez-le pour qu'il affiche en fin de programme le nombre de valeurs saisies et la somme.
- Calculer la valeur moyenne et affichez-là. Que constatez-vous ?
- Quelle solution proposez-vous ?
- Modifier le programme pour calculer uniquement la moyenne des éléments pairs. On rappelle que l'opérateur % calcule le reste de la division entière: par exemple 7%2 retourne 1, puisque 7 = 3*2 + 1.

Exercice 4

Objectifs : Ecrire un programme qui calcule la factorielle d'un **entier** quelconque. Utilisation d'une boucle for décroissante ou d'un test.

On rappelle que la factorielle est une fonction qui à un entier positif N associe :
N! = N*(N-1)*(N-2)*…*2*1, par convention 0! = 1

- Ecrivez un programme qui calcule N! d'un entier positif quelconque (0 compris).
- Calculez avec votre programme 8! (ou 17! suivant les réglages du compilateur). Que constatez-vous ? Quelle solution proposez-vous pour corriger cette erreur ?

Exercice 5

Objectifs : Programme qui calcule la moyenne de N **entiers**, utilisation de **variables dimensionnées.**

- Saisir le nombre de valeurs
- Saisir les valeurs dans une boucle.
- Calculer la somme des éléments du vecteur dans une autre boucle.
- Affichez la valeur moyenne

Exercice 6 : Moyenne

Objectif : Utiliser une fonction typée simple.

Reprenez le programme qui calcule la moyenne avec un tableau.

- Ecrivez la fonction moyenne avec le prototype suivant :

```
float moy(float somme , int N);
```

Cette fonction calcule la valeur moyenne des éléments d'un vecteur.

- Modifiez la fonction main( ) pour qu'elle utilise moy().
- Conclusion ?

Exercice 7 : Fonction et variable dimensionnée

O**bjectif** : Passer un tableau dans une fonction

➡ Ecrire une fonction saisie avec le prototype suivant:

```
void saisie(float A[], int N);
```

➡ Dans le main, affichez les valeurs saisies et calculer la somme

➡ En utilisant la fonction écrite dans l'exercice précédent, calculer la moyenne

Exercice 8 : synthèse

O**bjectif** : Passer un tableau dans une fonction

➡ Ecrire une fonction somme avec le prototype suivant:

```
float som(float A[], int N);
```

Cette fonction réalisera la somme des N réels stockés dans le tableau A

➡ Dans la fonction main, remplissez le tableau A grâce à une fonction appelée saisie, calculez la somme des éléments de A grâce à une fonction appelée som, et calculez la moyenne des éléments grâce à une fonction appelée moy.

Exercice 9 : Factorielle (facultatif)

Reprenez le programme qui calcule la factorielle.

➡ Ecrivez la fonction fact( ) avec le prototype suivant :

```
float fact( int N );
```

Cette fonction calcule la valeur de factorielle N.

➡ Modifiez la fonction main( ) pour qu'elle utilise fact( ).

II : Mini Projets

Objectifs :

Ce premier mini-projet a pour objectif de vous faire travailler des algorithmes avec plusieurs boucles imbriquées et de vous faire manipuler des fonctions sans paramètres de sortie.

Exercice "sapins"

Objectif : Ecrire un programme qui trace à l'écran un "sapin", le but étant la mise en œuvre d'algorithmes à boucles imbriquées.

- Ecrivez un programme qui écrive sur N lignes, d'abord une étoile sur la première ligne, puis 2 sur la ligne suivante etc. Autant d'étoiles que le numéro de la ligne en cours. (Fig 1)
- En modifiant le programme précédent, dessinez un triangle composé d'étoiles. (Fig 2)
- Ajoutez maintenant un tronc de 3 étoiles sur deux lignes à votre arbre. (Fig 3)

Fig 1	*Fig 2*	*Fig 3*
*	*	*
**	***	***
***	*****	*****
****	*******	*******
*****	*********	*********
		***
		***

Exercice "mon sapin en couleur"

Objectif : Ecrire un programme qui trace à l'écran un "sapin", le but étant la découverte des séquences d'échappement.

Insérez dans votre programme les définitions suivantes :

```
#define def "\033[0m" //couleur par defaut
#define high "\033[1m" //Highlight
#define soul "\033[4m" //Souligne
#define blink "\033[5m" //Clignotement
#define noir "\033[30m" //noir
#define rge "\033[31m" //rouge
#define vert "\033[32m" //vert
#define jaune "\033[33m" //Jaune
#define bleu "\033[34m" //bleu
#define pour "\033[35m" //pourpre
#define cyan "\033[36m" //cyan
#define blanc "\033[37m" //blanc
```

A partir de cet instant, il suffit d'envoyer au terminal la séquence d'échappement choisie pour modifier les paramètres d'affichage. Ceci se fait à l'aide d'un simple printf. : `printf("%s",rge);` bascule le texte d'affichage en rouge.

➡ Tracez votre sapin avec un feuillage vert et un tronc noir.

Exercice "mon sapin avec des boules clignotantes"

Objectif : Etudier un générateur de nombre aléatoire.

L'ordinateur est une machine déterministe. Rien n'est aléatoire dans son fonctionnement. Pourtant nous allons lui demander de placer les navires de façon aléatoire. Pour ce faire, nous allons utiliser la fonction **rand**. Cette fonction retourne un nombre de type **int** aléatoire entre 0 et **RAND_MAX**. **RAND_MAX** est une constante définie dans le header **stdlib.h**.

➡ Ecrivez un programme qui calcule et affiche 10 nombres aléatoires à l'aide de la fonction **rand**. Relancez plusieurs fois votre programme. Que constatez-vous ?

Pour éviter ce problème, il existe une fonction qui initialise le générateur de nombres aléatoires. Cette fonction **srand** initialise la série de nombres aléatoires générés par **rand**. Bien entendu, on pourrait saisir au clavier une valeur que l'on choisirait arbitrairement afin d'initialiser le générateur de nombres aléatoires, mais ce serait fastidieux.

Pour initialiser ce générateur, on utilisera le seul aléa que l'ordinateur subisse : l'heure de son allumage. Pour ce faire on utilise la fonction **time** qui est contenue dans le header **time.h** comme suit :

```
time_t t;
// on crée une variable t de type time_t (defini dans time.h)
srand(time(&t));
// on initialise le generateur de nombres aleatoires.
```

➡ Modifiez le programme précédent pour qu'il calcule 10 nombres aléatoires à l'aide de la fonction **rand** initialisée par **srand**. Relancez plusieurs fois votre programme. Que constatez-vous ?

Une fois le générateur initialisé, il faut restreindre les valeurs tirées par le générateur. En effet **RAND_MAX** vaut : 2 147 483 647. Pour restreindre ce nombre à N valeurs (mettons 100) on utilise la fonction **modulo** (**%**). En effet, quelque soit X, X%N est compris entre 0 et N-1 puisqu'il s'agit du reste de la division entière.

➡ Modifiez le programme précédent pour qu'il calcule 10 nombres aléatoires entre 0 et 10 à l'aide de la fonction rand initialisée par srand et de l'opérateur %.

On se propose de placer des boules dans le sapin, les boules seront figurées par des caractères '@'. On ne veut pas qu'il y ait trop de boules aussi limitera-t-on leur pourcentage tau (0% < tau < 100%), en empêchant la saisie hors de l'intervalle.

➡ Lors du placement de chaque élément du sapin on tirera un nombre aléatoire compris entre 0 et 100. Si ce nombre est <tau, alors on place un boule en mode clignotant sinon un élément de feuillage "*'. Bien entendu, pas de boules sur le tronc…

➡ On compliquera le code pour que le feuillage soit vert, les boules d'une couleur aléatoire.

Mini projet : La bataille navale.

Objectif : Réaliser un programme “bataille navale”. Afin de vous faciliter la tâche une analyse du problème vous est proposée. Respectez les étapes proposées et vous réussirez à réaliser ce programme en 2/3 séances.

Analyse sommaire du problème :

On réalisera le programme en respectant les consignes suivantes et en trois étapes. Un des problèmes essentiels est le choix du codage des données. Comment représenter le plateau de jeu dans la mémoire de l'ordinateur ?

Ensuite, il faut définir une représentation à l'écran du plateau de jeu. Pour des raisons de temps, vous ferez un quadrillage à base de caractères.

Il faut ensuite que l'ordinateur place les bateaux. Ceci va vous amener à étudier et utiliser le générateur de nombres aléatoires.

Enfin, vous allez gérer le jeu. Saisir les tirs du joueur, gérer le fait qu'il a pu toucher un navire en affichant un message approprié, gérer le fait qu'il a pu couler un navire (= toutes les cases du navire touchées) en affichant un message approprié. Enfin, vérifier que tous les bateaux n'ont pas été touchés auquel cas la partie serait terminée.

Codage des données :

Les données nécessaires au jeu seront codées de la façon suivante :

Le plateau est une variable dimensionnée à 2 dimensions de type **int**. Le contenu de chaque case peut être le suivant :

Le tableau contient 4 types de valeurs.

0 : On n'a pas testé cette case

x, et x>0 : La case est occupée par le bateau x

x, et x<0 : La case est occupée par le bateau x et il a été touché

999 : la case est vide et elle a été testée.

Note : Pendant la phase de développement du programme, on affiche le plateau comprenant les bateaux avec leurs numéros. Par la suite, on affiche soit **rien** (ce qui signifie que la case n'a pas été testée), soit **O** (= tir dans l'eau), soit **X**(= touché ou coulé).

Vous utiliserez une variable dimensionnée qui contient autant d'éléments que de bateaux +1.

Chaque élément de cette variable contiendra le nombre de cases non touchées du bateau en question. Ainsi, si cette variable porte le nom **bato** et que le bateau 1 n'a pas été touché, **bato[1]** devra contenir la valeur 2 (bateau de 2 cases). Touché une fois, la variable **bato[1]** sera décrémentée1 (**bato[1] = bato[1]-1**).

Il est rappelé que le contenu des cases est soit un numéro de bateau (positif non touché, négatif touché) soit 0 soit 999.

Note : on ne peut pas utiliser de bateau numéro 0 puisque c'est le code case vide, on a donc obligatoirement le premier bateau qui porte le numéro 1.

Affichage de la grille de jeu

La fonction suivante est capable d'effacer l'écran (le terminal) en utilisant des séquences d'échappement (le détail importe peu).

```
void clrscr(void)
{printf("\x1b[2J \x1b[0;0H");
}
```

Vous l'inclurez à tous vos programmes. L'appel de **clrscr()** efface l'écran.

Pour le développement du programme, vous afficherez la grille du plateau de jeu avec un exemple de contenu initialisé à la déclaration comme suit :

```
#include <stdio.h>
#define dim 10

typedef int plato[dim][dim];

…

int main(void)
{plato a={{-10,5,7},{10,5},{999,0,999}};
 …
 return 0;
}
```

[1] incrémenter : augmenter le contenu d'une variable de 1, décrémenter c'est le diminuer de 1.

Ce qui doit conduire à la grille suivante :

```
    0   1   2   3   4   5   6   7   8   9
  +---+---+---+---+---+---+---+---+---+---+
0 I10 I 5 I 7 I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
1 I10 I 5 I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
2 I O I   I O I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
3 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
4 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
5 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
6 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
7 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
8 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
9 I   I   I   I   I   I   I   I   I   I   I
  +---+---+---+---+---+---+---+---+---+---+
```

La grille est dessinée avec des caractères +, - et I (i majuscule).

Procédez par étapes !

- Vous commencerez par faire afficher une grille vide. Regardez la grille, elle est une succession de lignes avec des motifs identiques. Utilisez des directives #define pour définir des motifs que vous utiliserez avec profit pour afficher la grille.
- De même la taille de la grille est définie dans une constante en début de programme (dim) cf ci-dessus.
- puis vous prévoirez l'affichage du contenu des cases.
- Enfin, vous ferez de ce code une fonction que vous appellerez **aff**. Réfléchissez-bien aux variables qu'il faudra passer à cette fonction.

Il ne faut pas hésiter à essayer votre code d'affichage pour voir où sont les erreurs, comment passer à la ligne etc.

Gestion des tirs sur les bateaux.

Analyse algorithmique

Pour être capable d'afficher si on a coulé tous les bateaux, on utilisera une variable **NbCases** qui contient le nombre total de cases non touchées et occupées par des bateaux du plateau de jeu. Chaque fois que le joueur touche un des bateaux, on décrémente cette variable. Par conséquent, lorsqu'elle vaut zéro, c'est que tous les bateaux ont été coulés.

En parallèle, une variable “coup” est incrémentée à chaque essai, ce qui permettra de calculer le taux de réussite.

On testera toujours avec l'exemple donné ci-dessus, en positionnant correctement le contenu des différentes variables afin de ne pas perdre de temps (pour l'instant on développe le programme, on jouera plus tard !)

Afin de pouvoir afficher le message “coulé”, on crée une variable dimensionnée appelée **bato[ ]** qui pour chaque navire **k** contient le nombre de cases (la longueur) du navire en question. La grille contient, elle, les numéros des navires à la position qui est la leur.

Ainsi, si un tir tombe sur une case contenant la valeur **k**, on décrémente **bato[k]** et on met **-k** dans la case en question. En même temps on diminue **NbCases**.

Dès que **bato[k]** atteint 0, on affiche le message "coulé".

Il ne doit alors plus y avoir de case du plateau de jeu contenant la valeur **k**, mais uniquement des cases contenant la valeur **-k**.

Réalisation

Le programme saisira le tir de l'utilisateur sous forme de deux coordonnées numériques entières (x, puis y).

Il affichera la nouvelle grille tenant compte de ce tir, le nombre de coups tirés ainsi que le nombre de cases de bateau non touchées restantes.

➡ Ecrivez une fonction gestir

```
int gestir(plato a, int bato[nbr] )
```

qui réalisera les actions suivantes :

- saisie des coordonnées du tir.

- détermination si un navire est touché.

Si oui :

1. mise à jour de la variable bato[ ],
2. affichage touché ou coulé
3. mise à jour du plateau.

Cette fonction **gestir** est typée, elle retourne 0 si aucun navire n'est touché, -1 sinon. Cela afin de mettre à jour la variable **NbCases** dans la fonction **main**. Dans **main** on teste si la variable **NbCases** est nulle, auquel cas la partie est finie.

Placement aléatoire des bateaux

On peut maintenant placer les navires aléatoirement en utilisant la fonction rand() (cf. Le sapin).

Toutefois il faut que le navire reste dans la grille. Tirer un point de départ n'est pas suffisant. L'algorithme suivant permet de placer des navires de façon aléatoire sans déborder du plateau.

Note : Pour simplifier on supposera que tous les bateaux ont la même longueur : 2 cases.

```
Pour le bateau k
```

```
            1/ On tire un point de départ, (rand et modulo)
            2/ On tire une direction : un nombre entre 1 et 4 avec la conven-
            tion 1 : le bateau part vers le haut, 2 vers la droite, 3 vers le
            bas, 4 vers la gauche
            3/ Peut le placer dans ce cas ? Il faut : rester dans la grille,
            et que les cases concernées soient vides.
            Si oui on passe au bateau suivant, sinon, retour en 1
        fin de pour
```

Attention ! On ne peut placer un bateau qu'à deux conditions

- la case existe bien

- elle est vide

➡ Ecrivez une fonction qui place **nbr** navires dans la variable de type **plato** qui lui est transmise. **nbr** sera définie en constante en début de programme.

Quatrième partie

En fait, il n'y a pas de quatrième partie, le programme est fini ! Il n'y a qu'à assembler les morceaux et supprimer l'affichage en mode débogage du plateau de jeu. C'est à dire que la fonction **aff** n'affiche plus le contenu des cases, mais seulement les tirs (à savoir **X** ou **O**)

On peut envisager des améliorations si l'on a du temps afin de vérifier les saisies de l'utilisateur, voire même prévoir des navires de tailles différentes en initialisant les cases de variable **bato** aux dimensions des navires à placer. Il suffit alors dans la fonction écrite au point 5 de vérifier que toutes les cases (on connaît la longueur du navire que l'on place) sont bien libres.

Enfin, on peut imposer le fait que les navires ne doivent pas être contigus. Attention ce n'est pas simple !