Programmation C
TP n° 3 : Tableaux et Structures, et un peu d'aléa

Les exercices indiqués avec une $*$ sont à rendre.

Générateurs Pseudo-Aléatoires Pour obtenir un nombre (pseudo) aléatoire en C, vous pouvez utiliser la fonction `int rand()` de la librairie `stdlib.h`, qui retourne un entier aléatoire compris entre 0 et `RAND_MAX` (qui vaut au moins 32767 et qui est une constante spécifiée dans `stdlib.h`). Il faut commencer par initialiser la suite de nombres aléatoires en appelant la fonction `void srand(unsigned seed)` (de la biliothèeque `time.h`) avec par exemple comme graine de départ l'heure courante : `srand(time(NULL))`. Ensuite, à chaque appel de la fonction `rand`, celle-ci retourne un nouveau nombre aléatoire.

```c
#include<stdlib.h>
#include<time.h>

int main(void){
  srand(time(NULL));  //initialisation de la suite de nombre aléatoires
  int x = rand();     //chaque appel génère un nombre entre 0 et RAND_MAX;
  int y = rand();
}
```

Exercice 1 : Commandes $(*)$

Dans cet exercice, on utilise les définitions suivantes :

```c
enum etat {
  VALIDEE, ENCOURS, EXPEDIEE
};
typedef enum etat etat;

struct commande {
  int num_com;
  int prix_exp;
  int prix_prod;
  etat etat_com;
};
typedef struct commande commande;
```

Une commande est donc une structure comprenant des champs pour : le numéro de la commande, son prix d'expédition, le prix du produit, et l'état de la commande (validée, en cours ou expédiée).

1. Écrire une fonction `void affiche_com(commande c)` qui affiche une description de la commande `c`. On utilisera un `switch` pour afficher proprement l'état.
2. Dans un main, définissez deux variable de type `commande`. Vous pouvez choisir les valeurs que vous voulez pour les champs mais :

  - pour la première, vous déclarerez d'abord la variable puis initaliserez les valeurs des champs une par une.
  - pour la seconde, vous utiliserez la syntaxe permettant d'initialiser les différents champs sur la même ligne que la déclaration.

  Testez ensuite la méthode `affiche_com` sur ces deux variables.

3. Écrire une fonction d'en-tête `commande com_alea(int num)` qui renvoie une commande aléatoire : son prix d'expédition sera choisi aléatoirement entre 1 et 20, sa valeur entre 1 et 2000, son état sera choisi aléatoirement, et son numéro sera l'entier passé en argument. On rappelle que les valeurs d'une enum sont en fait des `int` qu'on peut donc initialiser par des valeurs entières, 0, 1, 2...
4. Dans le `main`, créez un tableau de `NBC` commandes (où `NBC` est une constante définie en préambule par `#define`), dont chaque entrée sera initialisée avec la fonction `com_alea`. Vous testerez les fonctions des questions suivantes sur ce tableau.
5. Faire une fonction `void affiche_exp(commande t[], size_t taille)` qui affiche les commandes *expédiées* du tableau `t`, supposé de taille `taille`.
6. Faire une fonction `int nbr_en_cours(commande t[], size_t taille)` qui renvoie le nombre de commandes *en préparation* dans le tableau `t`.
7. Faire une fonction `int cout_validees(commande t[],size_t taille)` qui renvoie le coût total d'expédition des commandes *validées* dans le tableau `t`.
8. Considérons le code suivant :

```c
int main(void){
  commande c = {.prix_prod=100};
  change_prix(c,200);
  printf("%d\n",c.prix_prod);
}
```

   Pensez vous qu'il soit possible d'écrire une fonction de signature `void change_prix(commande c, int newprice)` telle que le code précédent affiche 200 ? Quelle nouvelle version de la fonction permettrait de modifier le champ prix de cette structure `c` ?
9. Pensez vous qu'on puisse créer une fonction de signature

```c
void expedie_tout(commande tab[], size_t n)
```

   telle que si `tab` est un tableau de commande de taille `n` (comme celui crée dans la question 4) alors l'instruction `expedie_tout(tab,n)` dans un `main` modifie le champ `etat_comm` de toutes les commandes pour le passer à `EXPEDIEE`

Exercice 2 : Polynômes (∗)

On veut représenter par une structure des polynomes à coefficients entiers de degré inférieur à 100. On définira donc une constante N par `#define N 100` en début de code.
La structure sera la suivante :

```c
struct polynome{
  int de;
  int co[N];
};
```

L'idée est que le champ `co` est le tableau de coefficients du polynôme : le polynôme $3X^3 + 4X + 1$ sera représenté par une structure `p` dont le champ `p.co` sera le tableau `{1, 4, 0, 3,0,.....0}` (notez bien que le tableau a bien toujours taille `N`, quel que soit le degré du polynome).
La structure contiendra aussi un champ `p.de` qui sera le degré du polynome, c'est à dire l'indice maximal `i` tel que `p.co[i]` est non nul. Bien sûr, on pourrait ne pas stocker cette information dans la structure et la recalculer à chaque fois par un parcours du tableau, mais on peut imaginer que ce soit rentable de ne pas refaire ce parcours à chaque fois, quitte a perdre juste 4 octets de memoire sur la structure polynome.

1. Écrivez une fonction qui prend un polynôme ainsi qu'un entier et retourne le résultat de l'évaluation de ce polynôme. Pour l'exemple `p` ci dessus, `eval(p, -1)` retournera la valeur `-6`.
2. Écrivez une fonction `print_poly` qui affiche un polynôme sous sa représentation habituelle. Pour l'exemple donné en introduction, on voudra donc voir s'afficher `3X^3+4X+1`.
3. Écrivez une fonction `derive` qui prend un polynome et qui construit et retourne le polynome dérivé. Toujours avec l'exmple introductif, on doit donc retourner la structure correspondant au polynome $9X^2 + 4$.
4. Écrivez une fonction qui prend deux polynômes et retourne un nouveau polynôme résultant de leur addition. L'addition de deux polynômes s'effectue en sommant les coefficients de même degré. Par exemple, la somme de $1 + X + 2X^2$ et de $2 + 3X$ donne $3 + 4X + 2X^2$. Attention au bon calcul du degré.
5. Écrivez une fonction qui prend deux polynômes P et Q, et retourne un nouveau polynôme résultant de leur multiplication. Par exemple, la multiplication de $1 + X + 2X^2$ et de $2 + 3X$ donne $2 + 5X + 7X^2 + 6X^3$.

Exercice 3 : Calcul approché de π par méthode de Monte Carlo (Exercice BONUS)

1. Écrire une fonction qui prend en argument les coordonnées(x, y) d'un point du plan et retourne 1 s'il appartient au disque de rayon 1 et 0 sinon
2. Si on tire au hasard un point du plan à l'intérieur du carré $[0, 1] \times [0, 1]$, la probabilité qu'il appartienne au quart de disque de rayon 1 centré en l'origine est égal précisément à la surface de ce quart de disque. Un algorithme de calcul approché de la surface consiste donc à tirer plein de points au hasard dans le carré et de compter le nombre de points qui tombent dans le disque.

   Écrire une fonction qui prend en argument un entier n et calcule une valeur approchée de π en effectuant n tirages selon cet algorithme pour calculer la surface du quart de disque