Exercice 1:

Soit p et p1 deux variables déclarées comme suit :

```
#include <stdio.h>
int main (void)
{
    int *p;
    double *p1;
}
```

- Ecrire un programme C qui afficherait la taille mémoire occupée par p et p1.
- Que faut-il conclure ?

Exercice 2:

Compléter la ligne 8 du programme suivant (en remplissant les ....). Ce programme convertit une lettre majuscule (resp. minuscule) en une lettre minuscule (resp. majuscule). On suppose que l'utilisateur saisit soit une lettre majuscule soit une lettre minuscule.

```
1  #include <stdio.h>
2  int main (void)
3  {
4      char car;
5      printf ("\nMerci d'introduire une lettre majuscule ou minuscule: ");
6      scanf ("%c", &car);
7      printf ("\nla conversion (majuscule vs minuscule) de %c
8        donne %c.\n ", ..., ... + (....?  'a' -'A' : ...));
9      return(0);
10 }
```

Exercice 3:

Qu'affiche le programme C suivant :

```
#include <stdio.h>
int main (void)
{
    int *p=(int *) 0x100a;
    double *p1=(double *) 0x200b;
    printf ("%d  \n",sizeof(p1) > sizeof(p));
    printf ("%p  \n",p+1);
    printf ("%p  \n",p1+1);
    return(0);
}
```

On suppose que sizeof(int *)=8, sizeof(double)=8 et que sizeof(int)=4.

Exercice 4:

- Écrire un programme en langage C qui lit un entier positif `n` et affiche sa forme binaire en utilisant uniquement des opérations bit à bit.

  Quelques remarques :

  - Dans un premier temps, il est demandé d'afficher la représentation binaire de `n` de manière inversée. Par exemple, si l'utilisateur entre `14`, sa représentation binaire est `1110`, mais le programme doit afficher `0111`. Il n'est pas demandé d'afficher les zéros non significatifs.
  - Si l'utilisateur ne saisit pas un entier valide, le programme doit afficher un message d'erreur et s'arrêter en renvoyant un code d'erreur `return 1`.

- On gardera en tête que `n` est un entier non signé. Chaque fois qu'une constante entière est utilisée dans une expression impliquant la variable `n` (par exemple `0` ou `1`), elle devra être écrite avec le suffixe `u` pour indiquer qu'il s'agit d'une constante de type `unsigned int`.
- L'affichage des bits doit se faire exclusivement avec l'instruction `putchar()`. L'utilisation de `printf()` n'est pas autorisée pour l'impression des bits (sauf éventuellement pour le message d'erreur ou pour la demande de saisie d'un entier positif). Une fois tous les bits affichés, le programme doit ajouter un retour à la ligne également via `putchar`.

Exercice 5:

- Ecrire une fonction C, appelée miroir, qui prend en paramètre un entier et retourne son entier miroir. Par exemple, si le paramètre est égal à 3425 la fonction retournera 5243.

Exercice 6:

Écrire un programme en C qui lit, caractère par caractère, une séquence de bits se terminant par un retour à la ligne, puis convertit cette séquence en un entier positif. Nous utiliserons les opérations bit à bit pour résoudre cet exercice.
Par exemple, si l'utilisateur entre la séquence 1110 (d'abord 1, puis 1, puis 1 et enfin 0), le programme affichera le nombre 14.
Le programme affichera une erreur si un caractère différent de "0" ou "1" est rentré.

Exercice 7:

On a demandé à un étudiant d'écrire un programme C qui :

- lit un nombre positif ou nul n qui est inférieur ou égal à 8 (on lui a demandé d'utiliser la boucle do ... while pour avoir la bonne valeur de n), et
- affiche la valeur lu.

L'étudiant a écrit le programme suivant :

```
#include <stdio.h>
int main (void)
{

    int n;
    do {
        printf("\nMerci d'introduire un nombre positif ou nul :");
        n=scanf("%d", &n);
    } while ((n>8) || (n<0));
    printf ("Le nombre lu est : %d\n", n);
     return(0);
}
```

L'étudiant pense que son programme n'est pas juste car en rentrant les valeurs entières (par exemple, 3, 5, 4, 45) le programme affiche la valeur "1". Par ailleurs, lorsqu'un caractère est saisi (au lieu d'un entier attendu), le programme affiche la valeur "0".

Pouvez-vous commenter ces résultats et corriger le programme de l'étudiant.

Exercice 8:

Écrire un programme en C qui lit un ensemble de lettres minuscules et génère tous les sous-ensembles possibles, puis les affiche.

- L'ensemble en entrée est lu caractère par caractère, commençant par '{', suivie de lettres minuscules consécutives à partir de 'a' et séparées par des virgules, se terminant par '}'.
- Pour simplifier, les lettres sont supposées se suivre de manière consécutive.
- L'utilisation de tableaux n'est pas autorisée, mais les opérateurs bit à bit peuvent être utilisés pour mémoriser et manipuler les données.
- Nous supposons que l'utilisateur respecte bien le format en entrée

Exemple :

Si l'utilisateur entre l'ensemble `{a, b, c}`, le programme devra afficher les sous-ensembles suivants (l'ordre d'affichage des sous-ensembles n'est pas pertinent) :

Sous-ensembles
{}
{a}
{b}
{c}
{a, b}
{a, c}
{b, c}
{a, b, c}

Exercice 9:

Ecrire un programme C qui réalise le jeu du juste prix. Il s'agit de deviner, au bout d'un certain nombre d'essais, le prix d'un produit.

La valeur mystère du juste prix est générée aléatoirement. Pour cela, utiliser les deux instructions suivantes :

```
srand(time(NULL));
variable = rand() ;
```

rand () retourne une valeur entière positive que l'on peut borner grâce à l'opérateur modulo %. Ces deux instructions nécessitent d'inclure les bibliothèques suivantes :

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

Exercice 10 :

En mémoire, les mots mémoires (souvent représentés sur un octet) ne sont composés que de 0 et de 1.

On vous demande d'écrire une fonction qui imprime l'espace mémoire (en hexadécimal) associé à un nombre flottant. L'objectif est de vérifier si la norme IEEE-754 est utilisée pour représenter les nombres réels.

Par exemple, le nombre réel 227.625 a comme représentation 0 1000 0110 1100 0111 0100 0000 0000 000 (ou encore 4363A000 en héxadécimal) avec la norme IEEE-754 avec une simple précision (32 bits).

Exercice 11:

On se donne une table (que l'on appellera baguenaudière) à n cases, chacune peut contenir un pion. Chaque case est numérotée de "1" à "n". La baguenaudière peut être vide ou pleine. Le but du jeu est de remplir la baguenaudière si elle est vide, ou de la vider si elle est pleine, en respectant les règles suivantes :

- Pour la case 1, on peut enlever un pion ou mettre un pion sans contrainte.
- Pour la case 2, on peut enlever un pion ou mettre un pion uniquement si la case 1 est pleine
- De manière générale, pour la case i (différente de 1 et 2), on peut enlever un pion ou mettre un pion si :
  - La case $i-1$ est pleine (contient un pion)
  - Toutes les cases de 1 à $i-2$ sont vides.

Ecrire les fonctions remplir et vider. On utilisera la récursivité croisée pour écrire vos fonctions.