Important : Les programmes doivent être écrits en C. Une grande importance sera apportée à la clarté, la lisibilité des raisonnements et des programmes.

Remarque : On pourra toujours librement utiliser une fonction demandée à une question précédente, même si cette question n'a pas été traitée. On suppose que les librairies standards `stdbool.h`, `stdio.h` et `stdlib.h` sont déjà importées et vous n'avez pas besoin de le préciser.

Exercice 1 (quelques fonctions) :

1. Écrire une fonction `int factorielle(int n);` prenant en argument un entier n et renvoyant $n!$ si $n \geq 0$ (et -1 sinon).
2. Écrire une fonction `int suplog(int n);` prenant en argument un entier n et renvoyant le plus petit entier d tel que $0 \leq n \leq 2^d$.
3. Écrire une fonction `float somme(float *tab, int n);` prenant en arguments un tableau de flottants de taille n, et renvoyant la somme de ses éléments.
4. Écrire une fonction `void initialisation(int *tab, int n);` prenant en arguments un tableau d'entiers de taille n, et modifie `tab` par effets de bords de telle sorte que sa case d'indice i contienne $i * i$.
5. Écrire une fonction `bool chercher(int *tab, int x, int n);` prenant en arguments un tableau d'entiers de taille n et un entier x, et renvoie `true` si `tab` contient x (et `false` sinon).
6. Écrire une fonction `int *zeros(int n);` qui renvoie une tableau d'entiers de taille n dont toutes les cases contiennent 0.

Exercice 2 (multiplication à la Russe) : On présente ici la multiplication à la Russe qui calcule le produit de deux facteurs. L'un des facteurs est appelé **multiplicande** et l'autre **multiplicateur** pour distinguer leurs rôles dans l'algorithme. La technique de multiplication dite russe consiste à répéter les étapes suivantes jusqu'à avoir un **multiplicateur** nul :

- faire la division euclidienne par 2 du **multiplicateur**, ce qui donne un quotient et un reste ;
- le **multiplicateur** prend comme valeur le quotient ;
- si le reste est 1, ajouter le **multiplicande** au résultat (qui aura été initialisé à 0) ;
- multiplier le **multiplicande** par 2.

Exemple : pour 13×238 les étapes sont :

Opération	Multiplicateur	Reste	Multiplicande	Résultat
	13	0	238	0
13/2	6	1	476	238
6/2	3	0	952	238
3/2	1	1	1904	1190
1/2	0	1	3808	3094

Donc $13 \times 238 = 3094$.

1. Calculer 69×135 sur votre feuille, en utilisant la multiplication à la Russe.
2. Écrire une fonction impérative (i.e. non récursive) `int mult_russe(int a, int b);` qui renvoie la valeur de $a \times b$ en utilisant la multiplication à la Russe.
3. Montrer que votre fonction termine.
4. (a) Donner un invariant de boucle qui garantit la correction de votre fonction.
   (b) Prouver cet invariant, ainsi que la correction de votre algorithme.

Exercice 3 (chaînes de caractères) : On rappelle qu'une chaîne de caractères est un tableau de caractères terminant par le caractère `'\0'`. Ainsi une chaîne de caractères de longueur 10 est du type `char chaine[11]`. La fonction `strlen` donne la longueur d'une chaine de caractères (sans le `'\0'`). Par exemple, `strlen(chaine)` renvoie 10.

Un **palindrome** est un mot qui se lit identiquement dans les deux sens (par exemple "kayak"). Considérons le code incomplet suivant :

```c
#include <string.h>

int main()
{
  char* str;
  str = (char*)malloc(20*sizeof(char));
  printf("Donner un mot.\n");
  scanf("%s", str);
  bool res = true;
  int n = strlen(str);
  for (int i=0; i <= /*** TODO ***/; i++)
  {
    if (/*** TODO ***/)
    {
      res = false;
    }
  }
  if (res)
  {
    printf("ceci est un palindrome\n");
  }
  else
  {
    printf("ceci n'est pas un palindrome\n");
  }
}
```

1. Remplacer les deux `/*** TODO ***/` (lignes 11 et 13) par les expressions adéquates pour que ce programme décide si le mot entré au clavier par l'utilisateur est un palindrome ou non.
2. Pourquoi dans le `scanf` de la ligne 8, il n'y a pas de symbole `&` devant `str` ?
3. Écrire une fonction `int longueur(char *s);` qui prend en entrée une chaîne de caractères `s` et qui renvoie sa longueur (sans utiliser la fonction `strlen`).

Exercice 4 (complexité) : Donner la complexité des fonctions suivantes en fonction de n :

```c
int f1(int n)
{
  int res = 2;
  for (int i=1; i<n/2; i++)
  {
    for(j=0; j<i; j++)
    {
      res++;
    }
  }
  return res;
}
```

```c
int f2(int n)
{
  int res = 0;
  for (int i=1; i<n; i=2*i)
  {
    res++;
  }
  return res;
}
```

Exercice 5 (pointeurs) : Considérons la suite d'instructions suivantes :

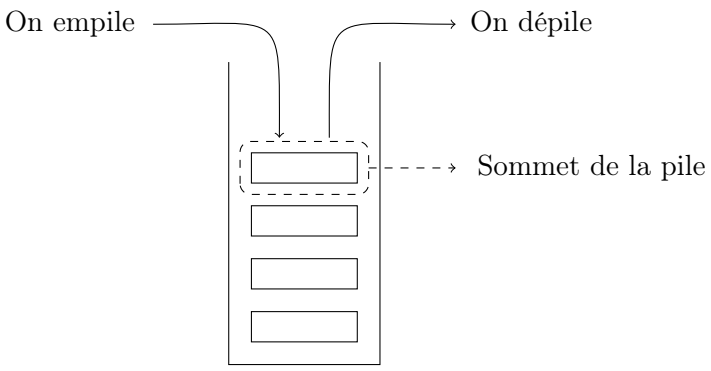
```
1 int x = 8;
2 int* p = &x;
3 int* p1 = p;
4 int y = x;
5 int* p2 = &y;
6 *p1 = 4;
7 p1 = p2;
```

Variable	Adresse	Valeur
x	100	
p	101	
p1	102	
p2	104	
y	103	
*p	—	
*p1	—	
*p2	—	

1. On suppose que les adresses des variables sont celles données dans le tableau ci-dessus. Compléter la colonne "Valeur" de ce tableau (une fois que toutes les instructions ont été exécutées).
2. Il est rarement intéressant d'initialiser un pointeur avec l'adresse d'une variable déjà existante. Rappeler la fonction de la librairie standard utile pour initialiser un pointeur.

Exercice 6 (piles) : On rappelle qu'une pile est une structure de données abstraite fondée sur le principe "dernier arrivé, premier sorti", ou LIFO ("last in, first out").
On suppose qu'on dispose d'une structure de pile déjà implémentée, capable de stocker des entiers. Ainsi, on dispose d'un type `pile` et des fonctions suivantes :

```
1 pile *creer_pile(); // crée une nouvelle pile vide
2 bool pile_est_vide(pile *p); // teste si p est la pile vide
3 void empiler(pile *p, int val); // empile val au sommet de p
4 int sommet(pile *p); // renvoie le sommet de p (p doit être non vide)
5 int depiler(pile *p); // dépile l'élément au sommet de p (non vide) et le renvoie
```



Une pile

1. On suppose dans cette question que la variable `p` contient une pile dont le contenu est le suivant (les éléments étant empilés par le haut) :

Que contiennent les variables `p` et `q` après l'exécution du code ci-dessous ?

```
1 pile *q = creer_pile();
2 while (!pile_est_vide(p))
3 {
4   empiler(q, depiler(p));
5 }
```

2. On appelle **hauteur** d'une pile le nombre d'éléments qu'elle contient.
   (a) Écrire une fonction `int hauteur_pile(pile *p);` qui prend en entrée un pointeur vers une pile et renvoie sa hauteur.
   Important : à la fin de votre fonction, la pile prise en entrée doit avoir retrouver son état d'origine.
   (b) Quelle est la complexité temporelle de votre programme pour une pile de taille n ?
3. Écrire une fonction `int max_pile(pile *p, int i);` telle que `max_pile(*p, i)` renvoie la position `j` du plus grand élément parmi les `i` derniers éléments de la pile `p`. Par convention, la position du sommet de la pile est 1.
   Important : à la fin de votre fonction, la pile `p` doit avoir retrouver son état d'origine.

Exemple

```
1 /* p contient la pile de la question 1 */
2 printf("%d\n", max_pile(p, 2));
```

Output

```
1
```

4. Écrire une fonction `void retourner(pile *p, int j);` telle que `retourner(p, j)` modifie `p` par effets de bords en inversant l'ordre des `j` derniers éléments empilés dans `p` et ne renvoie rien.
   Exemple : si `p` contient la pile de la question 1, alors après l'instruction `retourner(*p, 3)`, l'état de la pile `p` sera :

5. L'objectif de cette question est de trier une pile de crêpes.
   On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de pile) à la plus petite (placée au sommet de la pile).
   On dispose uniquement d'une spatule que l'on peut insérer dans le pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au dessus.
   Le principe est le suivant :
   - on recherche la plus grande crêpe ;
   - on retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile ;
   - on retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas ;
   - la plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

   (a) Écrire une fonction `void tri_crepes(pile *p);` qui trie la pile prise en entrée et ne renvoie rien.
   (b) Quelle est la complexité temporelle de votre programme ?