I31 – TD3

Exercice 1.1.
Définir ce qu'est un bloc d'instructions en langage C. Comment un bloc est-il matérialisé dans un programme C ?

Un bloc d'instructions est une structure permettant de grouper un ensemble d'instructions. En langage C, un bloc d'instructions est délimité par l'accolade ouvrante { pour le début du bloc et par une accolade fermante } pour la fin du bloc. Un bloc d'instruction peut être utilisé dans de nombreux endroits d'un programme. L'exemple suivant illustre un bloc en langage C :

```
{
  float f = 1.0f;
  printf("Dans le bloc: i = %d, f = %f", i, f);
}
```

Exercice 1.2.
Soit le programme suivant :

```
#include <stdio.h>

void main() {
  int i = 0;

  {
      float f = 1.0f;
      printf("Dans le bloc: i = %d, f = %f", i, f);
  }

  printf("Apres le bloc: i = %d, f = %f", i, f);
}
```

Que se passe-t-il lors de la compilation ? Lors de l'exécution ? En déduire le fonctionnement des blocs sur les déclarations de variables.

La compilation du programme précédent provoque une erreur de compilation à la ligne 10 (le second `printf`) indiquant que la variable `f` n'existe pas. Lors de l'utilisation d'un bloc d'instruction, toute variable déclarée à l'intérieur d'un bloc d'instructions n'existe qu'à l'intérieur de celui-ci et est détruite à la fin du bloc. Une variable déclarée dans un bloc d'instructions ne peut donc pas être réutilisée après le bloc. En revanche, les modifications réalisées à l'intérieur d'un bloc d'instructions sur des variables déclarées avant le bloc restent valident une fois le bloc terminé.

2. Structures conditionnelles

Exercice 2.1.
Qu'est-ce qu'une structure conditionnelle ? Ecrire en langage C les deux structures algorithmiques suivantes :

- **si** *condition* **alors** *instruction* **finsi**
- **si** *condition* **alors** instructionA **sinon** instructionB **finsi**

Une structure conditionnelle est une structure de contrôle qui permet de sélectionner des instructions à effectuer en fonction d'une condition. En langage C, la structure **si** *condition* **alors** i*nstruction* **finsi** est représentée par :

```
if (condition)                                          if (condition) {
  instruction ;         Si plusieurs instructions :       instruction ;
                                                        }
```

La structure **si** *condition* **alors** i*nstructionA* **sinon** instructionB **finsi** est représentée par :

```
if (condition)                                          if (condition) {
  instructionA;         Si plusieurs instructions :       instructionA;
else                                                    } else {
  instructionB;                                           instructionB;
                                                        }
```

Exercice 2.2.
Ecrire un programme qui affiche le minimum de 3 nombres entrés au clavier.

```
#include <stdio.h>

void main() {
    int a, b, c, min;

    scanf("%d %d %d", &a, &b, &c);

    if (a<b)
      min = a;                              // Si une seule instruction, pas besoin des {}
    else
      min = b;

    if (c<min)                              // Le min des 3 est le min du min des 2 premier et du 3ieme
      min = c;

    printf("Le nombre minimum est %d", min);
}
```

Exercice 2.3.
Comment représenter en langage C la structure conditionnelle algorithmique :

si *condition1* **alors**
instruction1
sinonsi *condition2* alors
instruction2
sinon
instruction3

En langage C, la structure conditionnelle précédente s'écrit :

```
if (condition1) {
  instruction1;
} else if (condition2) {
  instruction2 ;
} else {
  instruction3 ;
}
```

Exercice 2.4.
Ecrire un programme en langage C qui saisit un nombre au clavier et qui affiche :

- Multiple de 7 si le nombre est divisible par 7
- Pair si le nombre n'est pas divisible par 7 mais est pair
- Quelconque si le nombre n'est ni divisible par 7 ni pair.

```
#include <stdio.h>

int main()
{
    int i = 0;

    scanf("%d", &i);

    if (i % 7 == 0){
        printf("Divisible par 7");
    } else if (i % 2 == 0){
        printf("Pair");
    } else {
        printf("Quelconque");
    }

    return 0;
}
```

Exercice 2.5.
Décrire la syntaxe et le fonctionnement de la structure conditionnelle `switch` du langage C.

La structure conditionnelle switch permet d'exécuter du code selon la valeur d'une variable. Sa structure est la suivante :

```
switch(var){
  case constante1 : instruction1;
  case constante2 : instruction2;
}
```

Chaque instruction `case` vérifie si la valeur de `var` est égale à la constante correspondante. Si c'est le cas, la ou les instructions associées au `case` sont exécutées. Une fois qu'un `case` a été exécuté, tous les `case` suivants le sont également, et ce même si la valeur de `var` n'est pas égale à leurs constantes. Pour faire en sorte que seul le `case` correspondant à la valeur de `var` soit exécuté, il faut utiliser l'instruction `break` à la fin de chaque suite d'instruction d'un `case` :

```
switch(var){
  case constante1 : instruction1; break;
  case constante2 : instruction2; break;
}
```

La structure switch admet également une condition de défaut, qui est déclenchée systématiquement à la fin de la structure. Qu'un `case` ait été exécuté ou non. En revanche, si un `case` contient une instruction `break` et qu'il est exécuté, le `default` sera ignoré.

```
switch(var){
  case constante1 : instruction1; break;
  case constante2 : instruction2; break;
  default: instructionD;
}
```

Exercice 2.6.
Ecrire un programme en langage C qui saisit un caractère au clavier et qui affiche "Voyelle" s'il s'agit d'une voyelle et "Consonne" sinon.

```
#include <stdio.h>

int main()
{
    char c;

    scanf("%c", &c);

    switch(c){
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':
        case 'Y': printf("Voyelle"); break;
        default: printf("Consonne");
    }

    return 0;
}
```

Exercice 2.7.
Décrire la syntaxe et le fonctionnement de l'opérateur ternaire ? du langage C.

L'opérateur ternaire du langage C est une façon concise d'écrire un **si** … **sinon**. En plus de la concision, comme son nom l'indique il s'agit d'un opérateur, ce qui permet de l'utiliser au sein d'expressions (au contraire du `if`). La syntaxe de l'opérateur ternaire est la suivante :

```
(condition) ?instructionVrai :instructionFaux ;
```

Exercice 2.8.
Ecrire un programme en langage C qui demande à l'utilisateur le nombre d'enfants qu'il possède et affiche :

- "1 enfant" si l'utilisateur entre 1
- "x enfants" si l'utilisateur entre un nombre plus grand que 1

Proposer deux versions du programme : l'une basée sur `if`, l'autre sur l'opérateur ternaire `?`.

Version if :

```
#include <stdio.h>

int main(){
   unsigned int e;

   scanf("%u", &e);

   if (e < 2){
      printf("%u enfant", e);
   } else {
      printf("%u enfants", e);
   }

   return 0;
}
```

Version ternaire :

Dans cette version, on utilise l'opérateur ternaire ? non pas pour choisir la chaine de caractère à afficher, mais pour modifier la chaine en rajoutant le caractère `'s'` à enfant si l'on a plusieurs enfants. Ceci est possible car ? est un opérateur, et peut donc être intégré à une expression.

```
#include <stdio.h>

int main(){
    unsigned int e;

    scanf("%u", &e);

    printf("%u enfant%c", e, (e<2)?0:'s');

    return 0 ;
}
```

3. Structures itératives

Exercice 3.1.
Ecrire trois versions d'un programme qui effectue la saisie de 5 entiers et calcule la somme totale des nombres (avec boucle `for`, `while`, `do ... while`).

Le tableau suivant présente de façon synthétique les 3 versions possibles. La première ligne et la dernière ligne du tableau rassemblent les instructions communes aux 3 programmes. Chaque cellule de la ligne du milieu présente une version différente de l'itération. Chaque programme est obtenu en concaténant les instructions de la première ligne, les instructions de la cellule du milieu sélectionnée et les instructions de la dernière ligne.

```
#include <stdio.h>

int main(){

  int nombre;
  int somme = 0;
```

```
for(int i = 0; i < 5; i++){
  printf("Nombre %d: ", i+1);
  scanf("%d", &nombre);
  somme += nombre;
    }
```

```
int i = 0;
  while(i < 5){
      printf("Nombre %d: ", i+1);
      scanf("%d", &nombre);

      somme += nombre;

      i++;
  }
```

```
int i = 0;
    do {
        printf("Nombre %d: ", i+1);
        scanf("%d", &nombre);

        somme += nombre;

        i++;
    } while(i < 5);
```

```
  printf("somme: %d\n", somme);

  return 0;
}
```

Exercice 3.2.
Ecrire un programme qui affiche la moyenne d'une suite d'entiers positifs entrés au clavier. On arrêtera la saisie quand le nombre -1 est entré.

```
#include <stdio.h>

int main(){

    int nombre;
    float somme = 0;
    int cpt = 0;

    // On initialise avec un premier nombre
    printf("Entrer un nombre (-1 pour arreter): ");
    scanf("%d", &nombre);

    // Si le premier nombre saisi est -1 rien ne se passe
    // sinon tant que le nombre saisi est different de -1
    // on fait la somme des nombres et on on incremente le compteur
    while(nombre != -1) {
      somme += nombre;
      cpt++;

      printf("Entrer un nombre (-1 pour arreter): ");
      scanf("%d", &nombre);
    }

    // La moyenne n'est valide que si un nombre a ete saisi (cpt != 0)
    printf("Moyenne des %d nombres: %f\n", cpt, (cpt!=0)?somme/cpt:0);

    return 0;
}
```

Exercice 3.3.
Expliquer le fonctionnement des mots clés `continue` et `break` lorsqu'ils sont utilisés dans une structure itérative (`for`, `while`, `do ... while`).

L'instruction `continue` arrête l'exécution de l'itération actuelle et continue à l'itération suivante (retour au début de l'itération). Dans le cadre d'un `for`, l'instruction continue exécute l'instruction d'incrémentation avant de revenir au début de l'itération (pour éviter les boucles infinies).

L'instruction `break` arrête immédiatement l'exécution de l'itération actuelle et termine la structure itérative (reprend l'exécution directement après la structure itérative.)

Exercice 3.4.
Ecrire un programme en langage C qui saisit des nombres tant que `-1` n'est pas entré et qui affiche uniquement les nombres pairs (utiliser `continue` et `break`).

```
#include <stdio.h>

int main(){

    int n = 0;

    while(n != -1) {

      scanf("%d", &n);

      // Si n = -1 on sort de l'iteration
      if (n == -1)
        break;

      // Condition vraie si n % 2 != 0, donc n impair
      // dans ce cas on retournbe au debut de l'iteration
      if (n % 2)
        continue;

      printf("%d\n", n);

    }
}
```

Exercice 3.5.
Ecrire un programme qui calcule x^n, où x est un nombre réel et n un entier, tous deux entrés au clavier. Ecrire deux version, l'une utilisant une boucle `for`, l'autre une bouche `while`.

Version `while`

```
#include <stdio.h>

int main()
{
    float x;
    int n;
    float xn = 1.0;

    printf("Entrer x: ");
    scanf("%f", &x);

    printf("Entrer n: ");
    scanf("%d", &n);

    // Cas d'une puissance positive
    if (n > 0){

       // n etant initialise, pas besoin de la premiere instruction du for
       while(n > 0){
        xn = xn * x;
        n--;
      }

    // Cas d'une puissance negative
    } else if (n < 0){

      // n etant initialise, pas besoin de la premiere instruction du for
      while(n < 0){
        xn = xn / x;
        n++;
      }

    // Cas de la puissance 0
    } else {
        xn = 1;
    }

    printf("xn = %f", xn);

    return 0;
}
```

Version for :

```
#include <stdio.h>

int main()
{

    float x;
    int n;

    float xn = 1.0;

    printf("Entrer x: ");
    scanf("%f", &x);

    printf("Entrer n: ");
    scanf("%d", &n);

    // Cas d'une puissance positive
    if (n > 0){

       // n etant initialise, pas besoin de la premiere instruction
       // du for
       for(;n > 0; n--){
        xn = xn * x;
      }

    // Cas d'une puissance negative
    } else if (n < 0){

      // n etant initialise, pas besoin de la premiere instruction
      // du for
      for(;n < 0; n++){
        xn = xn / x;
      }

    // Cas de la puissance 0
    } else {
        xn = 1;
    }

    printf("xn = %f", xn);

    return 0;
}
```

Ce programme peut être optimisé en utilisant le fait que les structures itératives comportent une condition. Les if du programme mrécédent peuvent donc être retirés et le programme devient :

```
#include <stdio.h>

int main() {

    float x;
    int n;

    float xn = 1.0;

    printf("Entrer x: ");
    scanf("%f", &x);

    printf("Entrer n: ");
    scanf("%d", &n);


    // On entre dans ce for que si n > 0
    for(;n > 0; n--)
        xn = xn * x;

    // On entre dans ce for que si n < 0
    for(;n < 0; n++)
        xn = xn / x;

    // si n = 0, aucun des for n'a ete active et xn vaut 1 par initialisation
    printf("xn = %f", xn);

    return 0;
}
```

Exercice 3.6.
Ecrire un programme qui dit si un nombre entré au clavier est premier ou non.
RAPPEL : Un nombre premier n est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs. Ces deux diviseurs sont 1 et n.

```
#include <stdio.h>

int main()
{

    int nombre;
    int diviseur = 2;

    printf("Entrer un nombre entier: ");
    scanf("%d", &nombre);


    while(diviseur < nombre){
        if (nombre%diviseur == 0){
            printf("%d nest pas premier car divisible par %d", nombre, diviseur);
            return 0;
        }

        diviseur++;
    }

    printf("%d est premier.", nombre);
    return 0;
}
```

Exercice 3.7.
Ecrire un programme qui demande la saisie d'un nombre strictement positif jusqu'à ce que l'utilisateur réponde bien à l'attente.

```
#include <stdio.h>

int main()
{
    int nombre;

    printf("Entrer un nombre strictement positif: ");
    scanf("%d", &nombre);

    while(nombre <= 0){
      printf("Entrer un nombre strictement positif: ");
      scanf("%d", &nombre);
    }

    return 0;
}
```