Programmation 1: Fondamentaux

- Opérations Logiques
- Structures conditionnelles

Opérations Logiques

Concept

De manière abstraite

- **Vrai**
- **Faux**

En pratique

- Le type booléen fournit les valeurs **true** et **false**

En langage C

- De manière générale tout ce qui **ne vaut pas zéro** est interprété comme **vrai**
- L'utilisation de **stdbool.h** fournit la prise en charge du type booléen

Opérateurs logiques

Les opérateurs logiques interviennent dans des expressions qui produisent un résultat logique exprimé par Vrai ou Faux

A > 10 **et** A <20

On trouve principalement trois opérateurs

OU est vrai si **au moins une** des deux opérandes à la valeur **vrai**

ET est vrai si **les deux** opérandes ont la valeur **vrai**

NON est **vrai quand** l'opérande à la valeur **faux**

En langage C

Les opérateurs logiques comportent deux caractères

Fonction	Type	En C	Exemple
OU	Binaire	||	key == 'X' || key == 'x'
ET	Binaire	&&	A >= 10 && B > C
NON	Unaire	!	!(A%3==0)

Opérateurs logiques

L'opérateur **not** calcule l'état inverse d'une variable ou d'une expression logique

a	! a
true	false
false	true

Opérateurs logiques

OU

Pour que l'expression soit évaluée **true**, il faut qu'au moins l'une des deux opérandes soit égale à **true**

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

Opérateurs logiques

ET

Pour que l'expression soit évaluée **true**, il faut que les deux opérandes soit égale à **true**

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

Opérateurs logiques

Priorité des opérateurs

1. **!**
2. **&&**
3. **||**

```
bool a, b, c;
// ..
if (!a || b && c ) {.. }
// is equivalent to
if (!(a) || (b && c)) {.. }
```

Actions liées à une condition

Structure Conditionnelle

Exemple 1:

On souhaite tester la parité d'un nombre entré au clavier et afficher le résultat du test.

- Comment faire ?

Exemple 1:

On souhaite tester la parité d'un nombre entré au clavier.

- La valeur en entrée sera un entier
- Affichage de « Pair » si divisible par 2
- Affichage de « Impair » dans le cas contraire

Donc,

- Si reste de la division entière par 2 = 0 alors le nombre est pair

Faut-il refaire le test pour savoir si le nombre est impair ?

- Non !

Exemple 1:

```
Début
    Entier : Valeur ← 0

    Faire
    Afficher « Entrer un nombre entier »
    Lire Valeur

    Si Valeur % 2 = 0
    Alors
        afficher Valeur « est un nombre pair »
    Sinon
       afficher Valeur « est un nombre impair »
    FinSi
    Fait
Fin.
```

Exemple 1: Le code C

```
Début
    Entier : Valeur ← 0

    Faire
    Afficher « Entrer un nombre entier »
    Lire Valeur

    Si Valeur % 2 = 0
    Alors
        afficher Valeur « est un nombre pair »
    Sinon
        afficher Valeur « est un nombre impair »
    FinSi
    Fait
Fin.
```

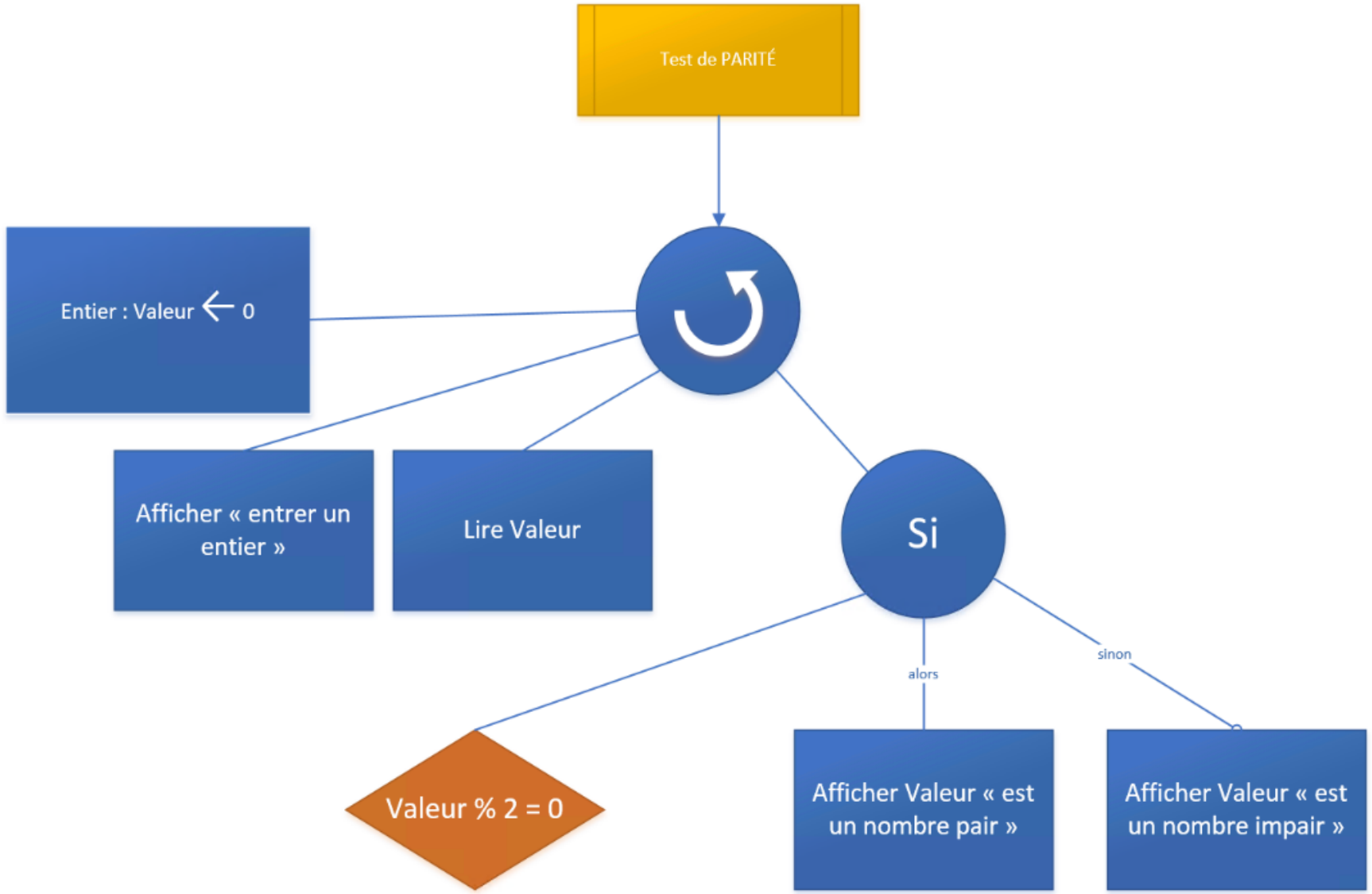
```
int valeur =  0;

printf("\nEntrer un nombre entier : ");
scanf_s("%d",&valeur);

if (valeur % 2 == 0)
        printf("%d est un nombre pair", valeur);
else
        printf("%d est un nombre impair", valeur);
```

Exemple 1:

Remarque sur ce pseudocode:

- On utilise une structure conditionnelle:
  Si test est VRAI alors action1 sinon action2
- Toutes les instructions ne sont donc pas exécutées, cela dépend du résultat du test.
- Le test est une expression qui doit pouvoir être évaluée en VRAI ou FAUX
- Action1 et Action2 peuvent aussi être des blocs d'instructions

Exemple 1:


Test de PARITÉ
Entier : Valeur ← 0
Afficher « entrer un entier »
Lire Valeur
Si
Valeur % 2 = 0
alors
sinon
Afficher Valeur « est un nombre pair »
Afficher Valeur « est un nombre impair »

Exemple 1:

Question bonus :

- A partir de quel test logique basé sur sa représentation binaire pourrait-on déterminer si un nombre entier est pair ou impair ?

Exemple 1:

Question bonus :

- A partir de quel test logique basé sur sa représentation binaire pourrait-on déterminer si un nombre entier est pair ou impair ?

Réponse :

```
// avec un test binaire sur le LSB
if ((valeur & 1) == 0)
printf("%d est un nombre pair", valeur);
else  printf("%d est un nombre impair", valeur);
```

Optimisation :

- Quelles parties du code semblent très proches et comment pourrait-on procéder à une factorisation de ce code?

```
Début
    Entier : Valeur ← 0

     Faire
     Afficher « Entrer un nombre entier »
     Lire Valeur

     Si Valeur % 2 = 0
     Alors
          afficher Valeur « est un nombre pair »
     Sinon
        afficher Valeur « est un nombre impair »
     FSi
     Fait
Fin.
```

Optimisation :

- Quelles parties du code semblent très proches et comment pourrait-on procéder à une factorisation de ce code?

```
Début
    Entier : Valeur ← 0

     Faire
     Afficher « Entrer un nombre entier »
     Lire Valeur
     afficher Valeur « est un nombre »
     Si Valeur % 2 = 0
     Alors
         afficher « pair »
     Sinon
       afficher « impair »
     FSi
     Fait
Fin.
```

```c
// factorisation du code redondant
printf("\nEntrer un nombre entier : ");
scanf_s("%d", &valeur);
printf("\n %d est un nombre ", valeur);
if (valeur % 2 == 0)
printf("pair");
else  printf("impair");
```

Exemple 2:

Algorithme de permutation de deux valeurs :

Les deux valeurs sont représentées par les variables A et B

Principe :

- On ne peut assigner directement la valeur de B à A, sinon on perd la valeur de A
- Il faut sauvegarder la valeur de A dans une troisième variable temporaire Tmp
- A reçoit alors la valeur de B
- B reçoit la valeur de Tmp

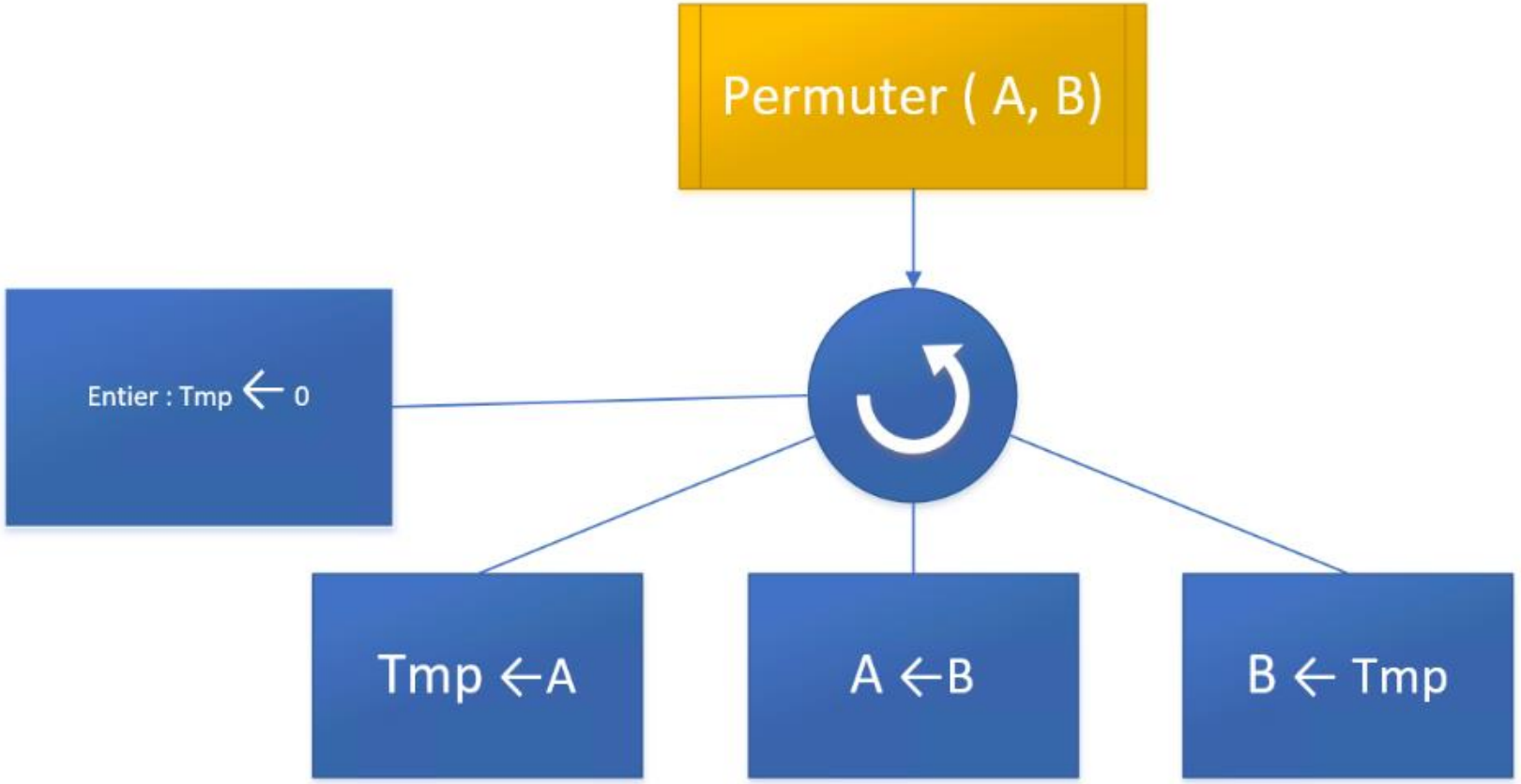
L'algorithme est le même qu'il s'agisse de valeur entières ou décimales, mais il faudra prévoir deux implémentations distinctes au moment du codage.

Exemple 2:

```
Début Permuter (Entier A, Entier B)
    Entier : Tmp

    Faire
            Tmp ← A
            A ← B
            B ← Tmp
    Fait
Fin Permuter.
```

```
int tmp, A = 10, B = 20;

tmp = A;
A = B;
B = tmp;
```

Exemple 2:



Exemple 2:

Algorithme de permutation de deux valeurs :

- Trouver un algorithme qui n'utilise pas de variable temporaire !
- Quel est l'inconvénient de cet algorithme lors de son implémentation ?

Exemple 2:

Début Permuter2 (Entier A, Entier B)

Faire

A ← A + B
B ← A - B
A ← A - B

Fait

Fin Permuter2.

```
// sans variable intermédiaire
A = A + B; // A contient la somme
de A et B
B = A - B; // B contient A
A = A - B; // A contient B
```

Cela ne marche qu'avec des variables numériques.
Il peut y avoir un dépassement de capacité dans certaines conditions (valeurs non entières)

Exemple 2:

Début Permuter3 (Entier A, Entier B)

Faire

A ← A XOR B
B ← A XOR B
A ← A XOR B

Fait

Fin Permuter3.

```
void XorSwap( int* x, int* y ) {
            if (x != y) {
                        *x ^= *y;
                        *y ^= *x;
                       *x ^= *y; }
             }
```

https://en.wikipedia.org/wiki/XOR_swap_algorithm

Cela ne marche qu'avec des variables numériques.
Il peut y avoir un dépassement de capacité dans certaines conditions (valeurs non entières)

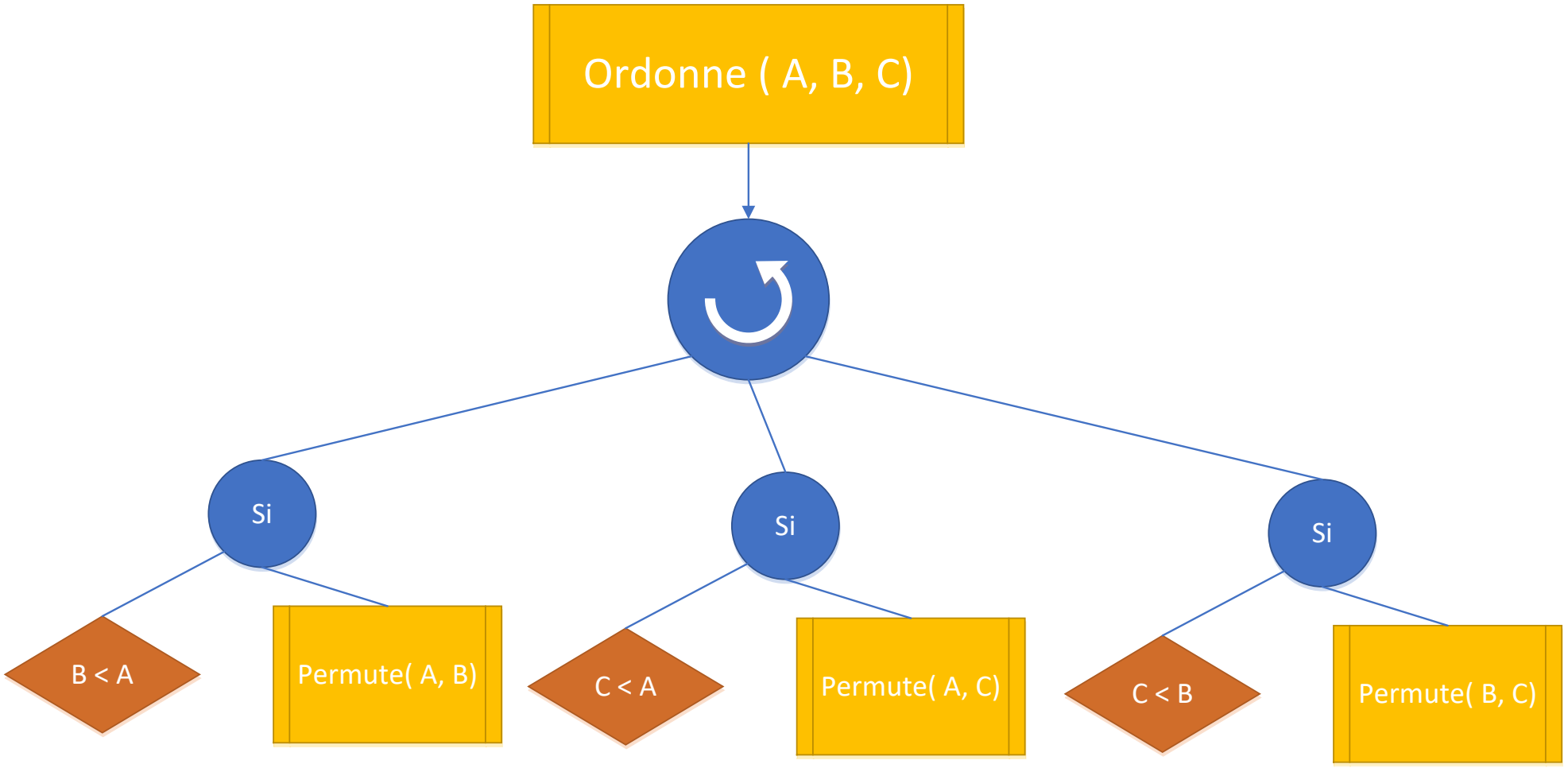
Exercice 1: Ordonner trois valeurs

Écrire un programme qui demande à l'utilisateur 3 valeurs entières A, B et C

Puis, range la plus petite dans la variable A , la suivante dans la variable B et la plus grande dans la variable C afin d'ordonner ces trois valeurs dans l'ordre croissant. Il peut y avoir des doublons dans les valeurs.

- L'algorithme ne débute le traitement qu'une fois qu'il connait les valeurs de A, B et C
- L'algorithme sera factorisé en faisant appel au code déjà écrit Permuter

Exercice 1: Ordonner trois valeurs



Exercice 1: Ordonner trois valeurs

Précisions :

- Dans l'arbre précédent, on utilise un Sous-programme déjà créé. Ce qui allège le dessin, mais surtout évite de réécrire toujours le même code. On factorise le code en limitant ainsi le risque d'erreurs et aussi la taille du programme.

- Les sous-programmes seront implémentés dans le langage cible à l'aide de procédures ou de fonctions

- Le formalisme utilisé ici ne fait pas apparaitre une notion importante pour la suite : les modifications apportées sur les variables par les sous-programmes doivent persister dans les programmes qui utilisent les sous-programmes. Nous reviendrons plus tard sur ces concepts.

Exercice 1: Ordonner trois valeurs

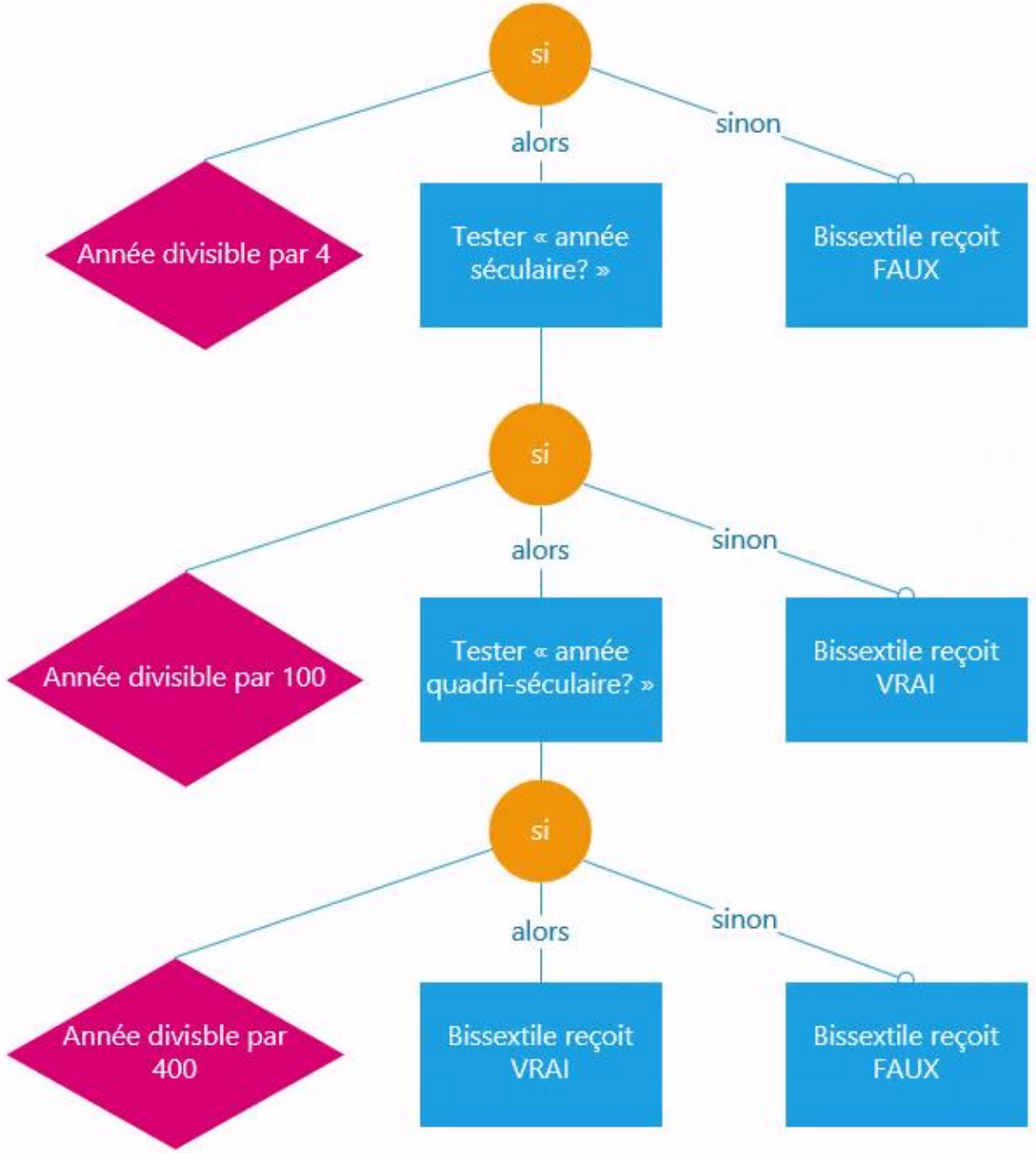
```
// Ordonner 3 valeurs
if (B < A) {
tmp = A; A = B; B = tmp;
}

if (C < A) {
tmp = A; A = C; C = tmp;
}

if (C < B) {
tmp = B; B = C; C = tmp;
}
```

Exercice 2:

Entrer un numéro d'année au clavier et afficher à l'écran si cette année est bissextile ou non. Ecrire seulement le **pseudocode** ou faire **un arbre programmatique**.

Définition du dictionnaire BIBLIOPLUS Larousse : Pour être bissextile, une année doit avoir son numéro divisible par 4. Toutefois celles dont le numéro est divisible par 100 ne sont bissextiles que si leur ce numéro est aussi divisible par 400 : 2000 était bissextile, 1700, 1800 et 1900 ne l'ont pas été.

Exercice 2: solution



Exercice 3:

Écrire un programme qui permet d'entrer au clavier le revenu net global d'un contribuable disposant d'une seule part fiscale, et qui calculera l'impôt brut.

Taux applicables aux revenus 2020 - Revenu imposable par part	
jusqu'à 10 064 €	0 %
de 10 065 € à 25 659 €	11 %
de 25 660 € à 73 369 €	30 %
de 73 370 € à 157 806 €	41 %
plus de 157 807 €	45 %

Exercice 3: Corrigé 1/3

```
// Calcul de l'impot en fonction du revenu net imposable

// déclaration de variables
int TB = 10065;double TTB = 0.11;
int TC = 25660;double TTC = 0.30;
int TD = 73370;double TTD = 0.41;
int TE = 157807; double TTE = 0.45;

int RNI = 0; // Revenu net imposable
double impot = 0; // montant de l'impôt

int revenuTmp;

// saisie du revenu net imposable
printf("\nEntrez votre revenu net imposable : ");
scanf_s("%d", &RNI);
revenuTmp = RNI;
```

Exercice 3: Corrigé 2/3

```
// montant dans la tranche E
if (revenuTmp > TE) {
impot = (revenuTmp - TE ) * TTE;
revenuTmp = TE;
}
// montant dans la tranche D
if (revenuTmp > TD) {
impot = impot + (revenuTmp - TD ) * TTD;
revenuTmp = TD;
}
// montant dans la tranche C
if (revenuTmp > TC) {
impot = impot + (revenuTmp - TC ) * TTC;
revenuTmp = TC;
}
// montant dans la tranche B
if (revenuTmp > TB) {
impot = impot + (revenuTmp - TB) * TTB;
revenuTmp = TB;
}
```

Exercice 3: Corrigé 3/3

```
// Affichage du montant de l'impôt

printf("\nle montant de votre impôt pour un revenu net imposable de %d
s'élève à %d euros.", RNI, (int)impot);
```

Exercice 4:

Ecrire un programme qui pour 3 nombres A, B et C entrés au clavier, étudie s'il existe un triangle A, B et C.

Pour que le triangle existe, il faut que chaque côté du triangle soit strictement inférieur à la somme des 2 autres côtés. Les 3 variables AB, BC et CA seront entrées au clavier après qu'un message nous y invite.

Exercice 4: corrigé 1/3

```
int AB = 0, BC=0, CA=0;

printf("\nEntrer la valeur du côté AB ");
scanf_s("%d", &AB);
printf("\nEntrer la valeur du côté BC ");
scanf_s("%d", &BC);
printf("\nEntrer la valeur du côté CA ");
scanf_s("%d", &CA);
```

Exercice 4: corrigé 2/3

```
// Version longue : imbrication des 'if'
int existe = 0; // par défaut , le triangle n'existe pas

if ((AB >= 0) && (BC >= 0) && (CA >= 0)) {
        if (AB + BC > CA) {
                if (AB + CA > BC) {
                        if (BC + CA > AB) {
                                existe = 1; // toutes les conditions sont vérifiées
                        }
                }
        }
}
if (existe)
printf("Ce triangle existe. \n");
else { printf("Ce triangle n'existe pas. \n"); }
```

Exercice 4: corrigé 3/3

```
// Version courte : Un seul 'if' regroupant les conditions
// avec des opérateurs logiques

if ((AB >= 0) && (BC >= 0) && (CA >= 0) && (AB + BC > CA)
                                        && (AB + CA > BC) && (BC + CA > AB))
        printf("Ce triangle existe. \n");
else
        printf("Ce triangle n'existe pas. \n");
```

Switch .. case

Usage

- Le Switch case permet de choisir le code à exécuter en fonction du résultat de l'évaluation d'une expression, la valeur d'une variable par exemple.

Syntaxe

```
switch (switch_on)
{
     case .. :
     case .. :
     ..
     default:
     break;
}
```

switch .. case

Exemple

```
int val=3;
switch (val) {
case 4:
       printf("\nNiv max");
       break;
case 3:
       printf("\nvalide niv 3");
case 2:
       printf("\nvalide niv 2");
case 1:
       printf("\nvalide niv 1");
break;
default: printf("non validé");
}
```

Le mot clé `break` permet de ne pas exécuter le code du reste des cas qui suivent le premier vérifié.