Structures Conditionnelles

En Algorithmique et en C

Sans conditions, nos programmes informatiques feraient toujours la même chose ! En programmation, les structures conditionnelles permettent de modifier le comportement d'un programme en fonction de certaines conditions.

Université Abou Bakr Belkaïd - Tlemcen
Faculté des Sciences - Département de Mathématiques
abourachanez@gmail.com

Introduction aux Structures Conditionnelles

On appelle structure conditionnelle les instructions qui permettent de :

Prendre une Décision

Le programme évalue une condition pour décider quelle séquence d'instructions exécuter ensuite.

Tester une Condition (Vrai / Faux)

Une expression logique qui, une fois évaluée, produit un résultat booléen : vrai (true) ou faux (false).

La Structure Conditionnelle Simple: (Si ... Alors / if ...)

L'instruction **if** est la structure de test la plus basique, on la retrouve dans tous les langages (avec une syntaxe différente...). Elle permet d'exécuter un bloc d'instructions **seulement si** une condition spécifique est vraie.

Pseudo-code (Algorithmique)

```
SI <condition> ALORS
        <instructions>
FINSI
```

L'exécution des instructions est conditionnée par la condition.

Langage C

```
if (<condition>) {
        // instructions
    }
```

Les accolades `{ }` délimitent le bloc d'instructions à exécuter si la condition est vraie.

Exemple Pratique:

Vérifions si un nombre est positif. Si c'est le cas, un message est affiché.

Pseudo-code (Algorithmique)

```
Algorithme nombre
Début
   x ← 5
  SI (x > 0) ALORS
        écrire ("x est positif.")
  FINSI
Fin.
```

☞ Dans cet exemple, le message "x est positif" sera affiché car la condition (`x > 0`) est vraie.

Langage C

```
int main() {
  int x = 5;
  if (x > 0) {
       printf("x est positif.\n");
                }
return 0;
 }
```

☞ L'instruction `printf` ne s'exécute que si la valeur de `x` est strictement supérieure à zéro.

La Structure Conditionnelle Composée:

(Si... Alors... Sinon / if ... else)

Permet d'offrir une alternative lorsque la condition initiale est fausse. Le programme exécute un bloc d'instructions ou un autre, sans exception.

Pseudo-code (Algorithmique)

```
SI <condition> ALORS
        <instructions_si_vrai>
    SINON
        <instructions_si_faux>
FINSI
```

👉 Une seule des deux branches d'instructions sera exécutée.

Langage C

```
if (<condition>) {
        // instructions_si_vrai
                }
    else {
        // instructions_si_faux
            }
```

👉 La clause else s’exécute dans tous les cas où la condition du if n’est pas vérifiée.

Exemple Pratique:

Déterminons si un étudiant est admis ou ajourné, basé sur sa note.

Pseudo-code (Algorithmique)

```
Algorithme admission
Début
  Lire (note)
  Si (note ≥ 10) alors
            Afficher ("Admis")
      Sinon
            Afficher ("Ajourné")
FinSi
Fin.
```

👉 Si `note` est **10 ou plus**, "Admis" est affiché.
Sinon, "Ajourné" est affiché.

Langage C

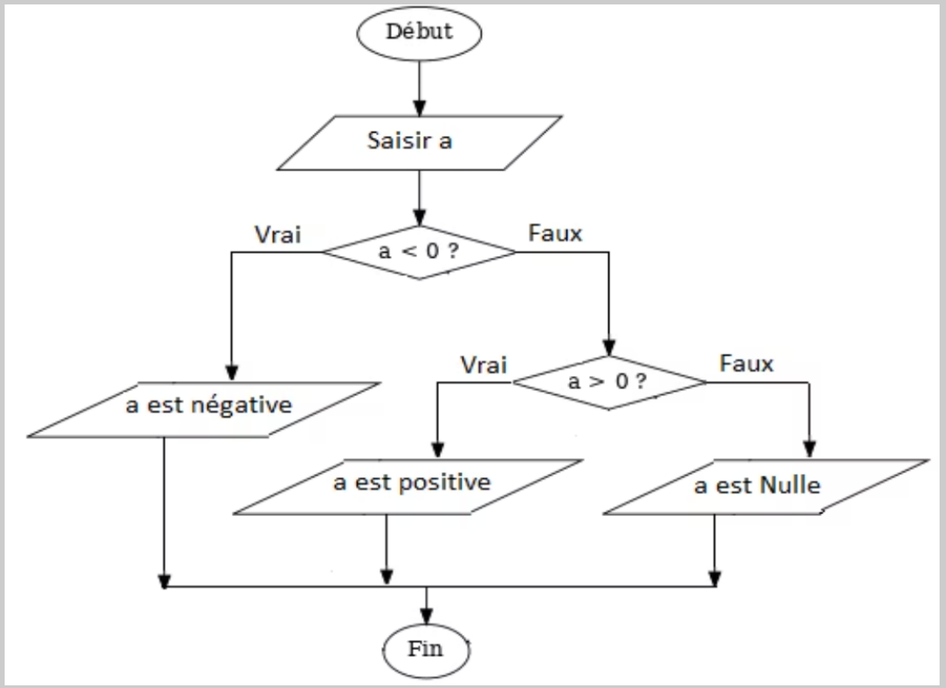
```
int main() {
  double note;
  printf("Entrez votre note:");
  scanf("%lf", &note);
  if (note >= 10) {
            printf("Admis.\n");
                        }
          else {
              printf("Ajourné.\n");
                 }
 return 0;
}
```

Structure conditionnelle multiple (Tests imbriqués) (if … else if … else)

Dans certains cas, une simple alternative **if…else** ne suffit pas. Lorsqu'un programme doit choisir entre plusieurs options possibles, on utilise la structure **if…else if…else**, qui permet de tester plusieurs conditions successive et d'exécuter l'action correspondant à la première condition vraie.

Exemple

Organigramme



Langage C

```
int main () {
int a = 0;
printf (" Saisie de a : ");
scanf ("%d" ,&a);

if  (a < 0)     printf ("la variable a est negative");
      else     {

            if (a > 0)    printf ("la variable a est positive");
                  else       // a est nulle
                        printf ("la variable a est nulle");
                  }
return 0;
  }
```

Opérateurs

Après avoir vu comment utiliser la structure conditionnelle `if`, il est important de savoir **comment formuler les tests** à l'intérieur de ces conditions. Pour cela, nous utilisons deux familles d'opérateurs

Les opérateurs de comparaison

Servent à comparer deux valeurs entre elles

Symbole	Signification
==	est égal à
>	est supérieur à
<	est inférieur à
>=	est supérieur ou égal à
<=	est inférieur ou égal à
!=	est différent de

Exemple

```
if (valeur1 <= valeur2)
                printf("%d >= %d\n", valeur2, valeur1);
    else    printf("%d > %d\n", valeur1, valeur2);
```

Les opérateurs logiques

Permettent de combiner plusieurs conditions

Symbole	Signification
&&	ET
||	OU
!	NON

Exemple

```
int x = 1, y = 0;
printf (" x || y = %d", x || y);
printf (" x && y = %d", x && y);
printf (" contraire (1) = %d" ,!(1));
```

Les Choix Multiples: Selon... Faire / Switch

Lorsque plusieurs cas distincts doivent être traités en fonction de la valeur d'une variable, l'imbrication de multiples instructions **if...else if...else** peut devenir lourde. les structures de choix multiples sont idéales car ils permettent de tester efficacement une seule expression contre plusieurs valeurs possibles.

- L'instruction **Switch** offre une alternative plus claire et plus lisible pour gérer plusieurs choix.

- Chaque valeur est associée à un **case**, et lorsque la valeur de l'expression correspond à celle d'un **case**, le bloc de code correspondant est exécuté.

- Le **default** sert à traiter tous les cas qui ne correspondent à aucune des valeurs précédentes.

Pseudo-code (Algorithmique)

```
SELON <variable> FAIRE
    CAS <valeur1> :
        <instructions1>
    CAS <valeur2> :
        <instructions2>
     ...
    CAS AUTRE :
        <instructions_par_defaut>
FIN SELON
```

Langage C (Switch)

```
switch (expression) {
    case valeur1:
        // instructions1
        break;      //Important!
    case valeur2:
        // instructions2
        break;
      ...
     default:
        // instructions_par_defaut
}
```

👉 L'instruction `break;` est cruciale pour sortir du `switch` après l'exécution du `case` correspondant.

Exemple Pratique:

Affichage du jour de la semaine en fonction d'un nombre (1 pour Lundi, etc.).

Code C

```
int main() {
  int jour = 3;         // Par exemple, Mercredi
  switch (jour) {
       case 1:   printf ("Lundi\n");    break;
       case 2:   printf ("Mardi\n");    break;
       case 3:   printf ("Mercredi\n");   break;
       case 4:   printf ("Jeudi\n");    break;
       case 5:   printf ("Vendredi\n");   break;
       case 6:   printf ("Samedi\n");     break;
       case 7:   printf ("Dimanche\n");   break;
       default:  printf ("Jour invalide\n");
                  }
 return 0;
}
```

Enchaînement des cas dans un `switch` : Erreur ou Intention ?

Rappel du rôle du `break`

- Le mot-clé `break` permet de **quitter** le `switch` après l'exécution d'un cas.
- Sans `break`, l'exécution **continue automatiquement** dans les cas suivants (*effet de chute*).

Exemple d'un oubli involontaire ⚠

```
int temperature = 0;
switch (temperature) {
      case 0:   printf("Eau solide\n");
      case 25:  printf("Eau liquide\n");
                break;
      case 100: printf("Eau gazeuse\n");
}
```

☞ L'absence de `break` après `case 0` provoque l'exécution du cas suivant, ce qui est une **erreur logique** ici.

➞ Résultat :

```
Eau solide
Eau liquide
```

Exemple d’un retrait volontaire du **break** (utilisation judicieuse) 💡

```
int jour = 6;           // 1=Lundi ... 7=Dimanche
switch (jour) {
      case 6:
      case 7:    printf("Week-end\n");
            break;
       default:   printf("Jour de semaine\n");
}
```

👉 Ici, **l’absence de** `break` **après** `case 6` est *volontaire*, car les deux cas (6 et 7) doivent exécuter le même traitement.

➞ Résultat :

```
Week-end
```

Conditions d'utilisation de `switch` :

Pour pouvoir utiliser `switch` en C, certaines **conditions doivent être respectées** concernant la variable testée :

1. **La variable testée doit être de type entier ou caractère**, c'est-à-dire :
   - `int`
   - `char`
   - ou tout type convertible en entier (comme `short`, `long`, etc.).

☞ On **ne peut pas** utiliser `float`, `double` ou `string`.

2. **Les valeurs des** `case` **doivent être des constantes** (pas des variables ni des expressions calculées).

```
int vitesse = 90;
  switch (vitesse) {

      case 50:  printf ("Vitesse normale en ville\n");      break;

      case 90:  printf ("Vitesse sur route\n");             break;

      case 120: printf ("Vitesse sur autoroute\n");         break;

      default:  printf ("Valeur inconnue\n");
  }
```

3. **Une seule variable est testée**

Contrairement à `if`, le `switch` ne permet pas de combiner plusieurs conditions logiques (comme `a > 5 && b < 10`).

Exemples interdits :

Code C

```
switch (x + y)
```
❌ Expression complexe

```
switch (note > 10)
```
❌ Condition booléenne

```
int prix = 100,   tva = 19;

 switch (prix) {
     case 19:   printf("Prix avec TVA fixe de 19%%\n");     break;

     case tva:  printf("Prix avec TVA variable\n");      break;

     default:   printf("Autre prix\n");
 }
```

`case 19:` ✅ **autorisé**, car `19` est une **valeur fixe écrite directement dans le programme**.

`case tva:` ❌ **interdit**, car `tva` est **une variable** dont la valeur est connue **seulement à l'exécution**.

👉 `tva` n'est pas une valeur fixe écrite directement dans le code, c'est une variable dont la valeur peut changer pendant l'exécution.

Opérateur conditionnel

L’opérateur conditionnel (opérateur ternaire) permet d’écrire de façon plus concise une affectation conditionnelle simple.

Code C

```
variable = (condition) ? valeur_si_vrai : valeur_si_fausse;
```

Exemples

```
*  (moyenne >=10) ? printf (" Admis ") : printf (" Ajourne ");
*  admis = (( moyenne >=10) ? 1 : 0);
```

☞ Il offre un moyen compact d’attribuer une valeur à une variable en fonction d’une seule condition, rendant le code plus clair et plus court dans les cas simples.

☞ Il ne s’agit pas d’une simple instruction, mais d’une expression qui renvoie une valeur.

Récapitulatif et Importance

La maîtrise des structures conditionnelles est indispensable pour concevoir des programmes capables de s’adapter dynamiquement aux différentes situations et aux données traitées.

Si... Alors (Simple)

Exécution optionnelle d'un bloc.

Si... Alors... Sinon (Composée)

Choix entre deux blocs d'instructions.

Choix Multiples (Switch)

Gestion de nombreux cas distincts.

👉 Ces outils vous permettent de créer des algorithmes capables de prendre des décisions et de s'adapter à différentes situations, une compétence cruciale en programmation.

👉 Rédiger un code conditionnel efficace nécessite le respect de bonnes pratiques et la connaissance des pièges potentiels :

Utilisez toujours des accolades

Même pour une seule instruction, les accolades `{}` améliorent la clarté et préviennent les erreurs logiques, surtout lorsqu'on ajoute d'autres lignes plus tard.

Priorisez la lisibilité

Évitez les conditions trop complexes ; décomposez-les si nécessaire. Un code clair est plus facile à comprendre et à déboguer.

Utilisez `switch` pour les valeurs discrètes

Lorsqu'il s'agit de tester une variable avec plusieurs valeurs distinctes, `switch` est souvent plus clair et plus efficace qu'une longue chaîne de `if...else if`.