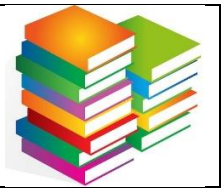
Base du langage C++ pour Arduino



Comment le langage C++ permet-il de décrire et d'organiser le fonctionnement d'un système piloté par une carte Arduino ?

Organisation d'un programme C++

Un programme est constitué d'instruction exécutée dans un ordre déterminé.

```
int compteur = 0;
compteur = compteur + 1;
```

Les instructions

Une instruction se termine par un point-virgule.

```
int temperature = 20;
temperature = temperature + 5;
```

L'oubli d'un point-virgule provoque une erreur lors de la compilation.

Les blocs d'instructions

Les accolades { et } regroupent plusieurs instructions dans un même bloc.

```
if (temperature > 25)
{
    etatVentilateur = true;
    compteur = compteur + 1;
}
```

L'indentation n'est pas obligatoire pour le compilateur, mais elle rend le programme beaucoup plus lisible.

Les commentaires

Les commentaires permettent d'expliquer le programme. Ils ne sont pas exécutés.

Commentaire sur une ligne :

```
// Déclaration de la température
int temperature = 20;
```

Commentaire sur plusieurs lignes :

```
/*
  Ce programme analyse
  la température du système.
*/
```

Les variables et les constantes

Une variable

Une variable est une zone de mémoire possédant :

- ⇨ Un nom
- ⇨ Un type
- ⇨ Une valeur qui peut évoluer

```
int compteur = 0;
```

Dans cette instruction :

Élément	Signification
`int`	Type de la variable
`compteur`	Nom de la variable
`0`	Valeur initiale

La valeur peut ensuite être modifiée :

```
compteur = 5;
compteur = compteur + 1;
```

Après ces deux instructions, la variable `compteur` contient la valeur `6`.

Une constante

Une constante possède une valeur qui ne doit pas changer pendant l'exécution du programme.

```
const byte LED_VERTE = 8;
```

Les noms des constantes sont généralement écrits en majuscules pour les repérer facilement.

```
const int SEUIL_TEMPERATURE = 30;
const byte NOMBRE_LED = 3;
```

Les principaux types de données

Le type indique la nature de l'information stockée.

Type	Utilisation	Exemple
`bool`	Valeur vraie ou fausse	`bool actif = false;`
`byte`	Petit entier positif	`byte broche = 8;`
`int`	Nombre entier	`int compteur = 15;`
`long`	Grand nombre entier	`long distance = 250000;`
`unsigned long`	Grand nombre entier positif	`unsigned long temps = 0;`
`float`	Nombre décimal	`float tension = 4.75;`
`char`	Un caractère	`char lettre = 'A';`

Les valeurs booléennes

Une variable de type `bool` ne peut prendre que deux valeurs :

```
true
false
```

Exemple :

```
bool systemeActif = false;

systemeActif = true;
```

Elle est particulièrement utile pour mémoriser l'état d'un système, d'un mode ou d'un voyant.

L'affectation et les calculs

L'opérateur d'affectation

Le signe `=` place une valeur dans une variable.

```
temperature = 25;
```

La variable `temperature` reçoit la valeur `25`. Attention, ce n'est pas une égalité mathématique.

```
compteur = compteur + 1;
```

Cette instruction signifie :

- ⇨ Lire la valeur compteur
- ⇨ Ajouter 1
- ⇨ Enregistrer le résultat dans compteur

L'instruction peut être raccourcie :

```
compteur ++ ;
```

Les opérateurs de calcul

Opérateur	Opération	Exemple
+	Addition	a+b
-	Soustraction	a-b
*	Multiplication	a*b
/	Division	a/b
%	Reste d'une division	a%b

Exemple :

```
int valeur = 17;
int reste = valeur % 2;
```

La variable `reste` vaut `1`. La valeur 17 est donc impaire.

Les opérateurs de comparaison

Une comparaison produit une valeur booléenne true ou false

Opérateur	Signification
==	Est égal à
!=	Est différent à
<	Est inférieur à
>	Est supérieur à
<=	Est inférieur ou égal à
>=	Est supérieur ou égal à

Exemple :

```
temperature > 30
```

Cette expression vaut :

- ⇨ `true` si la température est supérieur à 30
- ⇨ `false` dans le cas contraire

Attention à = et ==

```
temperature = 30;
```
La variable reçoit la valeur 30

```
temperature == 30;
```
Le programme vérifie si température est égale à 30

Les opérateurs logiques

Les opérateurs logiques permettent d'associer plusieurs conditions.

Opérateur	Signification
&&	ET
||	OU
!	NON

Opérateur ET

Les deux conditions doivent être vraies :

```
if (temperature > 20 && temperature < 30)
{
    etat = 1;
}
```

Le bloc est exécuté si `température` est comprise entre `20` et `30`.

Opérateur OU

Au moins une condition doit être vraie :

```
if (mode == 1 || mode == 2)
{
    autorisation = true;
}
```

Le bloc est exécuté si `mode` vaut `1` ou `2`.

Opérateur NON

L'opérateur `!` inverse une valeur booléenne

```
bool actif = true;

actif = !actif;
```

Après cette instruction, `actif` vaut `false`.

Cette écriture permet notamment de faire changer un état :

```
etatLed = !etatLed;
```

Les conditions

La structure if

La structure if exécute un bloc seulement si une condition est vraie.

```
if (temperature > 30)
{
    alarme = true;
}
```

Si la condition est fausse, le bloc est ignoré.

La structure if...else

```
if (temperature > 30)
{
    alarme = true;
}
else
{
    alarme = false;
}
```

Un seul des deux blocs est exécuté.

La structure if...else if...else

Elle permet de choisir entre plusieurs situations.

```
if (temperature < 15)
{
    etat = 0;
}
else if (temperature < 25)
{
    etat = 1;
}
else
{
    etat = 2;
}
```

Le programme teste les conditions dans l'ordre.

Pour une `température` de `10` :

- ⇨ `temperature` < `15` est vraie
- ⇨ le premier bloc est exécuté
- ⇨ les autres conditions ne sont pas testées

La structure switch...case

La structure switch...case permet d'exécuter des instructions différentes selon la valeur d'une variable.

```
switch (mode)
{
    case 0:
        etat = 10;
        break;

    case 1:
        etat = 20;
        break;

    case 2:
        etat = 30;
        break;

    default:
        etat = 0;
        break;
}
```

Fonctionnement

```
switch (mode)
```

Indique la variable à analyser

```
case 1:
```

Correspond à la situation où `mode` vaut `1`

```
break;
```

Permet de quitter la structure après l'exécution du bloc.

```
default:
```

Est exécuté si aucune valeur précédente ne correspond.

Les boucles

Une boucle permet de répéter plusierus fois un bloc d'instructions.

La boucle while

```
while (compteur < 5)
{
    compteur++;
}
```

Le bloc est répété tant que la condition est vraie.

Passage	Valeur avant exécution	Condition
1	0	Vraie
2	1	Vraie
3	2	Vraie
4	3	Vraie
5	4	Vraie
Sortie	5	Fausse

La variable compteur vaut finalement 5.

Risque de boucle infinie

```
while (compteur < 5)
{
    etat = true;
}
```

La valeur de `compteur` ne change jamais. La condition reste donc toujours vraie et le programme ne quitte pas la boucle.

La boucle for

La boucle for est utilisée lorsque le nombre de répétition est connu.

```
for (int i = 0; i < 4; i++)
{
    compteur++;
}
```

Elle contient trois éléments :

```
for (initialisation; condition; évolution)
```

Dans l'exemple :

```
int i = 0
```

Crée le compteur et lui donne la valeur initiale 0

```
i < 4
```

Est la condition de poursuite

```
i++
```

Augmente le compteur après chaque passage.

Le boc est exécutée quatre fois pour les valeurs de i 0, 1, 2 et 3.

Les fonctions

Une fonction regroupe plusieurs instructions sous un même nom.

Elle permet :

- ⇨ D'éviter de répéter du code
- ⇨ De rendre le programme plus lisible
- ⇨ De décomposer un programme complexe
- ⇨ De réutiliser un même traitement

Fonction sans argument et sans retour

```
void initialiserCompteur()
{
    compteur = 0;
}
```

Appel de la fonction :

```
initialiserCompteur();
```

Le mot `void` signifie que la fonction ne renvoie aucune valeur.

Fonction avec argument et sans retour

```
void modifierCompteur(int nouvelleValeur)
{
    compteur = nouvelleValeur;
}
```

Appel de la fonction :

```
modifierCompteur(5);
```

La valeur `5` est transmise à l'argument `nouvelleValeur`.

Fonction sans argument et avec retour

```
int obtenirCompteur()
{
    return compteur;
}
```

Appel de la fonction :

```
int valeur = obtenirCompteur();
```

La fonction renvoie un entier grâce à l'instruction `return`.

Fonction avec arguments et avec retour

```
int additionner(int valeur1, int valeur2)
{
    int resultat = valeur1 + valeur2;

    return resultat;
}
```

Appel de la fonction :

```
int somme = additionner(4, 3);
```

Déroulement :

- ⇨ `valeur1` reçoit `4`
- ⇨ `valeur2` reçoit `3`
- ⇨ la fonction calcul `7`
- ⇨ elle renvoie le `resultat`
- ⇨ la variable `somme` reçoit `7`

Portée des variables

Une variable déclarée en dehors des fonctions est une variable globale.

```
int compteur = 0;

void fonction1()
{
    compteur++;
}

void fonction2()
{
    compteur = 10;
}
```

Elle peut être utilisé par toutes les fonctions du programme.

Une variable déclarée dans uen fonctin ou dans un bloc est une variable locale.

```
void calculer()
{
    int resultat = 5;
}
```

La variable `resultat` n'existe que dans la fonction `calculer()`

```
void fonction1()
{
    int valeur = 10;
}

void fonction2()
{
    valeur = 20;  // Erreur : valeur n'existe pas ici
}
```

Les erreurs fréquentes

Oubli du point-virgule

```
int compteur = 0    // Erreur
```

Correction :

```
int compteur = 0;
```

Confusion entre affectation et comparaison

```
if (compteur = 5)    // Incorrect dans ce contexte
```

Correction

```
if (compteur == 5)
```

Accolage manquante

```
if (compteur > 2)
{
    compteur = 0;
```

Correction

```
if (compteur > 2)
{
    compteur = 0;
}
```

Majuscules et minuscules

Le langage C++ est sensible à la casse :

```
compteur
Compteur
COMPTEUR
```

Ces 3 noms sont différents.

Variable non initialisée

A éviter :

```
int compteur;
```

Il est préférable de donner immédiatement une valeur :

```
int compteur = 0;
```

Synthèse

Élément	Utilisation
Variable	Stocker une valeur modifiable
Constante	Stocker une valeur fixe
`if`	Exécuter selon une condition
`else if`	Tester une autre condition
`else`	Traiter les autres situations
`while`	Répéter un nombre défini de fois
`switch`	Choisir selon une valeur
Fonction	Regrouper et réutiliser des instructions
Argument	Transmettre une information à une fonction
`return`	Renvoyer une valeur