C Arduino - Les variables

Dans la programmation Arduino, les variables permettebt de stocker et manipuler des données. Dans le cadre de la robotique, les variables permettent de gérer les données provenant de capteurs, de contrôler les moteurs, ou encore de suivre l'état du robot. Dans cet article, nous allons explorer les différents types de variables en C pour Arduino, leur déclaration, assignation, et usage, avec des exemples pratiques adaptés à des projets robotiques.

Découverte

Qu'est-ce qu'une variable ?

Une **variable** est une zone de mémoire dans laquelle vous pouvez stocker une valeur qui peut changer au cours de l'exécution de votre programme. Les variables ont trois caractéristiques principales :

- **Nom** : l'identifiant de la variable (exemple : `vitesseMoteur`).
- **Type** : le type de données qu'elle peut contenir (exemple : entier, flottant, etc.).
- **Valeur** : la donnée actuelle stockée dans la variable.

Déclaration et assignation de variables

1. Déclaration

Déclarer une variable signifie informer le programme que vous souhaitez réserver de la mémoire pour cette variable, en précisant son type et son nom.

Syntaxe :

```
type nomVariable;
```

2. Assignation

Attribuer une valeur à une variable se fait grâce à l'opérateur `=`.

Syntaxe :

```
nomVariable = valeur;
```

Vous pouvez aussi combiner la déclaration et l'assignation en une seule étape.

Syntaxe combinée :

```
type nomVariable = valeur;
```

Types de variables en Arduino

Voici les principaux types de variables utilisés dans la programmation Arduino, avec des exemples adaptés à la robotique.

1. Les entiers (`int` , `unsigned int`)

Les entiers permettent de stocker des nombres entiers, positifs ou négatifs.

- **int** : stocke des valeurs de -32 768 à 32 767.
- **unsigned int** : stocke des valeurs positives uniquement (0 à 65 535).

Exemple : Contrôle de la vitesse d'un moteur.

```
int vitesseMoteur = 100; // Vitesse initiale du moteur
unsigned int distanceObstacle = 50; // Distance mesurée par un capteur en cm
```

2. Les flottants (`float` , `double`)

Les flottants sont utilisés pour stocker des nombres décimaux. Ils sont utiles pour les calculs de précision.

- **float** : précision jusqu'à 6-7 chiffres après la virgule.
- **double** : souvent identique à `float` sur Arduino (pas de différence).

Exemple : Calcul de la distance parcourue par un robot.

```
float distanceParcourue = 12.34; // En mètres
double angleRotation = 45.5; // En degrés
```

3. Les caractères (`char`)

Les caractères permettent de stocker une lettre ou un symbole ASCII. Ils sont souvent utilisés pour gérer des messages ou des commandes.

Exemple : Communication avec un robot via Bluetooth.

```
char commandeRecue = 'A'; // Commande reçue : avancer
```

4. Les chaînes de caractères (`char[]` ou `String`)

Les chaînes de caractères sont utilisées pour manipuler des mots ou des messages.

- **char[]** : tableau de caractères, plus optimisé pour les microcontrôleurs.
- **String** : classe de manipulation de chaînes, plus facile à utiliser mais consomme plus de mémoire.

Exemple : Affichage d'un message sur un écran LCD.

```
char messageLCD[] = "Robot prêt"; // Message statique
String messageDynamique = "Vitesse : 100"; // Message variable
```

5. Les booléens (`bool`)

Les booléens stockent des valeurs `true` (vrai) ou `false` (faux). Ils sont utiles pour gérer des états.

Exemple : Vérification de la détection d'un obstacle.

```
bool obstacleDetecte = false; // Initialisation
```

Bonnes pratiques pour utiliser les variables

1. **Noms explicites** : Donnez des noms significatifs à vos variables pour comprendre leur rôle.
    - Mauvais exemple : `int a;`
    - Bon exemple : `int vitesseMoteur;`
2. **Initialisation** : Assurez-vous que vos variables sont initialisées avant leur utilisation pour éviter des comportements imprévisibles.
3. **Limitation de la portée** : Déclarez vos variables au niveau approprié (locale ou globale) pour éviter les conflits et réduire la consommation de mémoire.
4. **Évitez les types coûteux** : Préférez des types simples (comme `char[]`) plutôt que `String` si la mémoire est limitée.

Variables globales et locales

Variables globales

Déclarées en dehors de toute fonction, les variables globales sont accessibles dans tout le programme.

Exemple : Stocker la vitesse du robot.

```
int vitesseGlobale = 100; // Accès depuis n’importe quelle fonction
```

Variables locales

Déclarées à l'intérieur d'une fonction, elles ne sont accessibles que dans cette fonction.

Exemple : Utiliser une variable temporaire pour un calcul.

```
void avancer() {
    int vitesseLocale = 120; // Variable spécifique à cette fonction
    analogWrite(3, vitesseLocale);
}
```

Exemples pratiques en robotique

1. Utilisation des capteurs et moteurs

Dans cet exemple, nous utilisons un capteur ultrasonique pour détecter un obstacle et ajuster la vitesse des moteurs.

```
int trigPin = 9;
int echoPin = 10;
int moteurGauche = 3;
int moteurDroit = 5;
unsigned int distanceObstacle;
int vitesseMoteur = 100;

void setup() {
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(moteurGauche, OUTPUT);
    pinMode(moteurDroit, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    // Mesure de la distance
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
```

```
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    distanceObstacle = pulseIn(echoPin, HIGH) / 58; // Conversion en cm

    // Ajustement de la vitesse en fonction de la distance
    if (distanceObstacle < 10) {
        vitesseMoteur = 0; // Stop si trop proche
    } else if (distanceObstacle < 30) {
        vitesseMoteur = 50; // Ralentit si obstacle à mi-distance
    } else {
        vitesseMoteur = 100; // Pleine vitesse si pas d'obstacle
    }

    // Commande des moteurs
    analogWrite(moteurGauche, vitesseMoteur);
    analogWrite(moteurDroit, vitesseMoteur);

    // Affichage des données
    Serial.print("Distance: ");
    Serial.print(distanceObstacle);
    Serial.print(" cm, Vitesse: ");
    Serial.println(vitesseMoteur);

    delay(100);
}
```

2. Suivi de ligne avec des capteurs infrarouges

Dans cet exemple, deux capteurs infrarouges détectent une ligne noire et ajustent les moteurs pour suivre cette ligne.

```
int capteurGauche = 2;
int capteurDroit = 4;
int moteurGauche = 3;
int moteurDroit = 5;
bool gaucheDetecte;
bool droitDetecte;

void setup() {
    pinMode(capteurGauche, INPUT);
    pinMode(capteurDroit, INPUT);
    pinMode(moteurGauche, OUTPUT);
    pinMode(moteurDroit, OUTPUT);
}

void loop() {
    gaucheDetecte = digitalRead(capteurGauche);
    droitDetecte = digitalRead(capteurDroit);

    if (gaucheDetecte && droitDetecte) {
        analogWrite(moteurGauche, 100); // Avance droit
        analogWrite(moteurDroit, 100);
    } else if (gaucheDetecte) {
```

```
        analogWrite(moteurGauche, 50); // Tourne à gauche
        analogWrite(moteurDroit, 100);
    } else if (droitDetecte) {
        analogWrite(moteurGauche, 100); // Tourne à droite
        analogWrite(moteurDroit, 50);
    } else {
        analogWrite(moteurGauche, 0); // Stop si hors ligne
        analogWrite(moteurDroit, 0);
    }

    delay(100);
}
```

Conclusion

Les variables sont au cœur de vos projets robotiques avec Arduino. Bien choisir leur type et leur portée, les initialiser correctement, et les utiliser de manière structurée est essentiel pour garantir la fiabilité et la clarté de votre code. Avec ces bases et les exemples fournis, vous êtes prêt à gérer efficacement les données pour construire des robots plus performants et intelligents.

Exercice boussole avec Wokwi

Pour vous exercer, effectuez l'exercice de la boussole en C arduino. pur ce fair, rendez-vous sur www.wokwi.com et sélectionnez un nouveau projet pi pico.

Voici le code et le schéma de base:

© 2025 Mr. Lambrechts https://v2.ttrinfo.be 7 | 7