ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

# Introduction au langage C

## Comprendre les Microcontrôleurs

Jean-Daniel NICOUD et Pierre-Yves ROCHAT

# Introduction au langage C

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

- Un peu d'histoire...
- Variables et assignations
- Structures de contrôle
- Procédures
  et programme principal

# Histoire du C

- Créé en 1972 par Dennis Ritchie  dans les laboratoires Bell d'AT&T.
- Résultat de l'évolution de de langages plus anciens

# Histoire du C

- Créé en 1972 par Dennis Ritchie  dans les laboratoires Bell d'AT&T.
- Résultat de l'évolution de de langages plus anciens
- Écriture concise, proche de l'assembleur
- Sa simplicité en fait un bon candidat pour les microcontrôleurs

# Histoire du C

- Créé en 1972 par Dennis Ritchie  dans les laboratoires Bell d'AT&T.
- Résultat de l'évolution de de langages plus anciens
- Écriture concise, proche de l'assembleur
- Sa simplicité en fait un bon candidat pour les microcontrôleurs
- Standardisation ANSI en 1989 et 1999
- Le C++ (orienté objet) est une très grande extension du C

# Histoire du C

- Créé en 1972 par Dennis Ritchie  dans les laboratoires Bell d'AT&T.
- Résultat de l'évolution de de langages plus anciens
- Écriture concise, proche de l'assembleur
- Sa simplicité en fait un bon candidat pour les microcontrôleurs
- Standardisation ANSI en 1989 et et 1999
- Le C++ (orienté objet) est un très grande extension du C
- GCC ( GNU Compiler Collection ) : base de beaucoup de compilateurs pour microcontrôleurs.

# Variables et assignations

- `int x` définit une **variable**
  (ici, un nombre, on parlera plus tard des types)

# Variables et assignations

- `int x` définit une **variable**
  (ici, un nombre, on parlera plus tard des types)

- L' assignation :

```
x = 12;
x = y + 3;
x = x - 2;
```

# Variables et assignations

- `int x` définit une **variable**
  (ici, un nombre, on parlera plus tard des types)

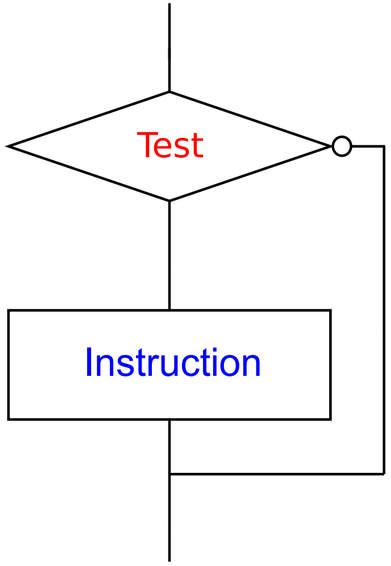
- L' assignation :

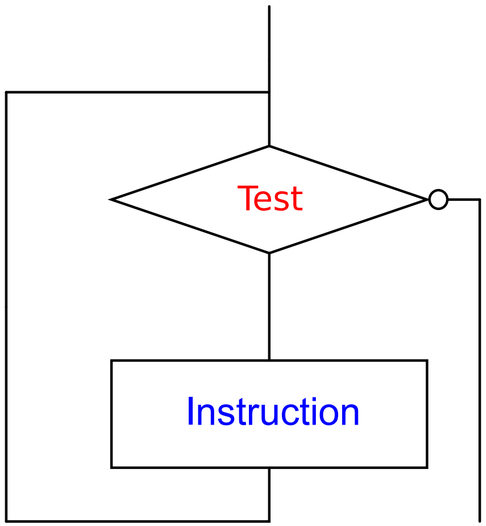
```
x = 12;
x = y + 3;
x = x - 2;
```

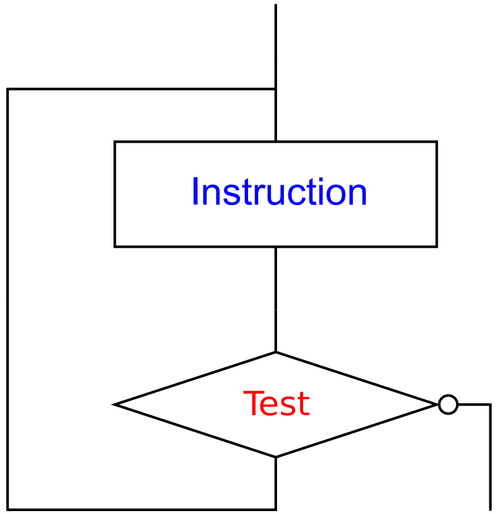
- Les opération arithmétiques : `+` `-` `*` `/` `%`

# Structures de contrôle

*Structures de contrôle de base :*



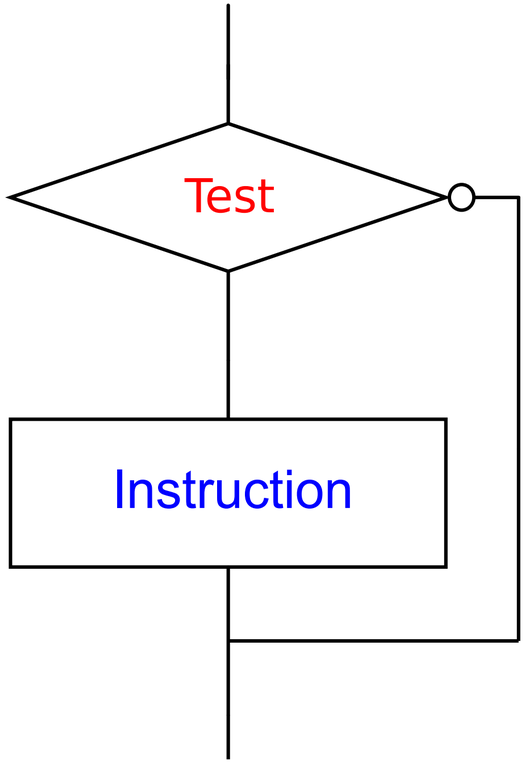




Test : `if`

Boucles : `while` `do...while`

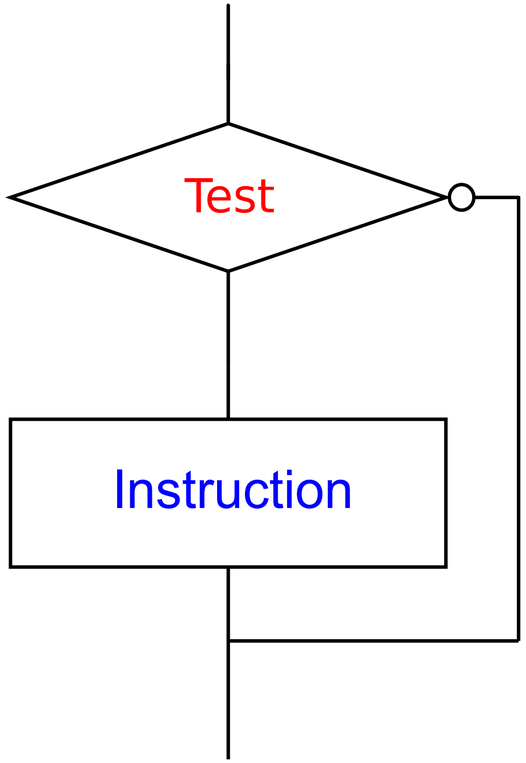
# if : le test de base




```
if (Condition)
{
   Instruction;
}
```

```
if (Condition){
   Instruction;
}
```

# if : le test de base

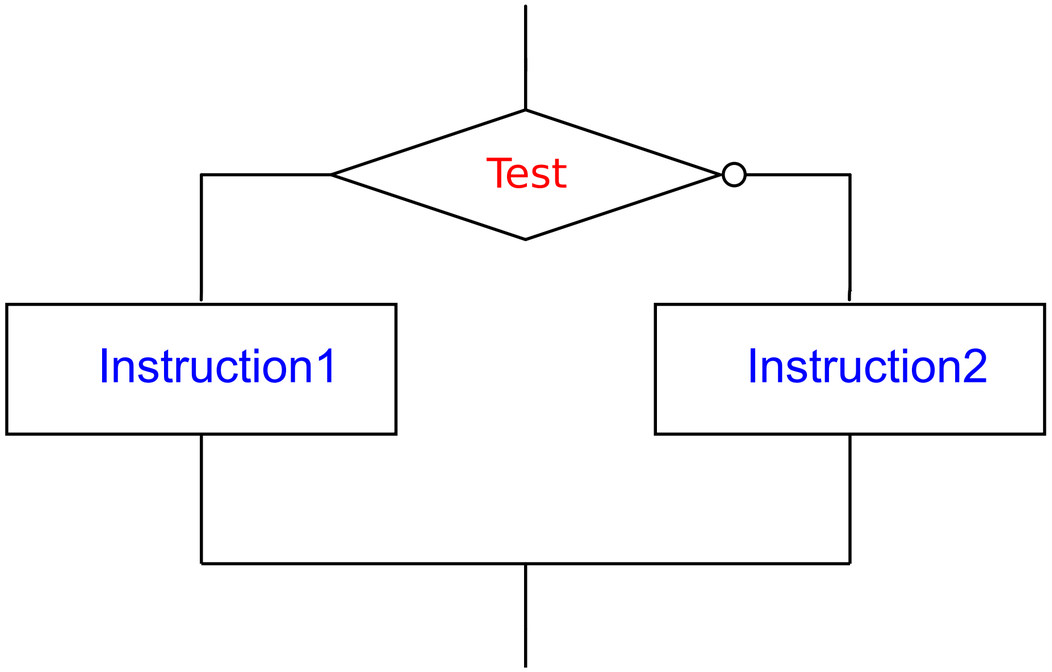



```
if (Condition)
{
   Instruction;
}
```

```
if (Condition){
   Instruction;
}
```

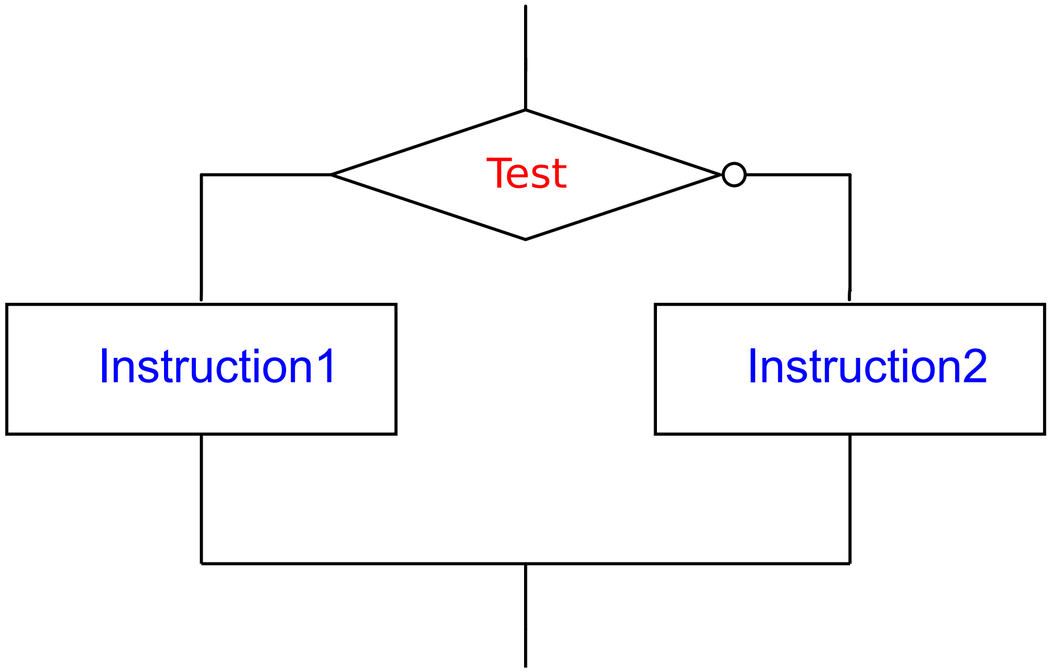
La structure `if` est un test, ensuite le programme se poursuit !
L'instruction peut s'exécuter 0 ou 1 fois.

# if...else : le test complet




```
if (Condition)
{
   Instruction1;
}
else
{
   Instruction2;
}
```

# if...else : le test complet




```
if (Condition)
{
   Instruction1;
}
else
{
   Instruction2;
}
```

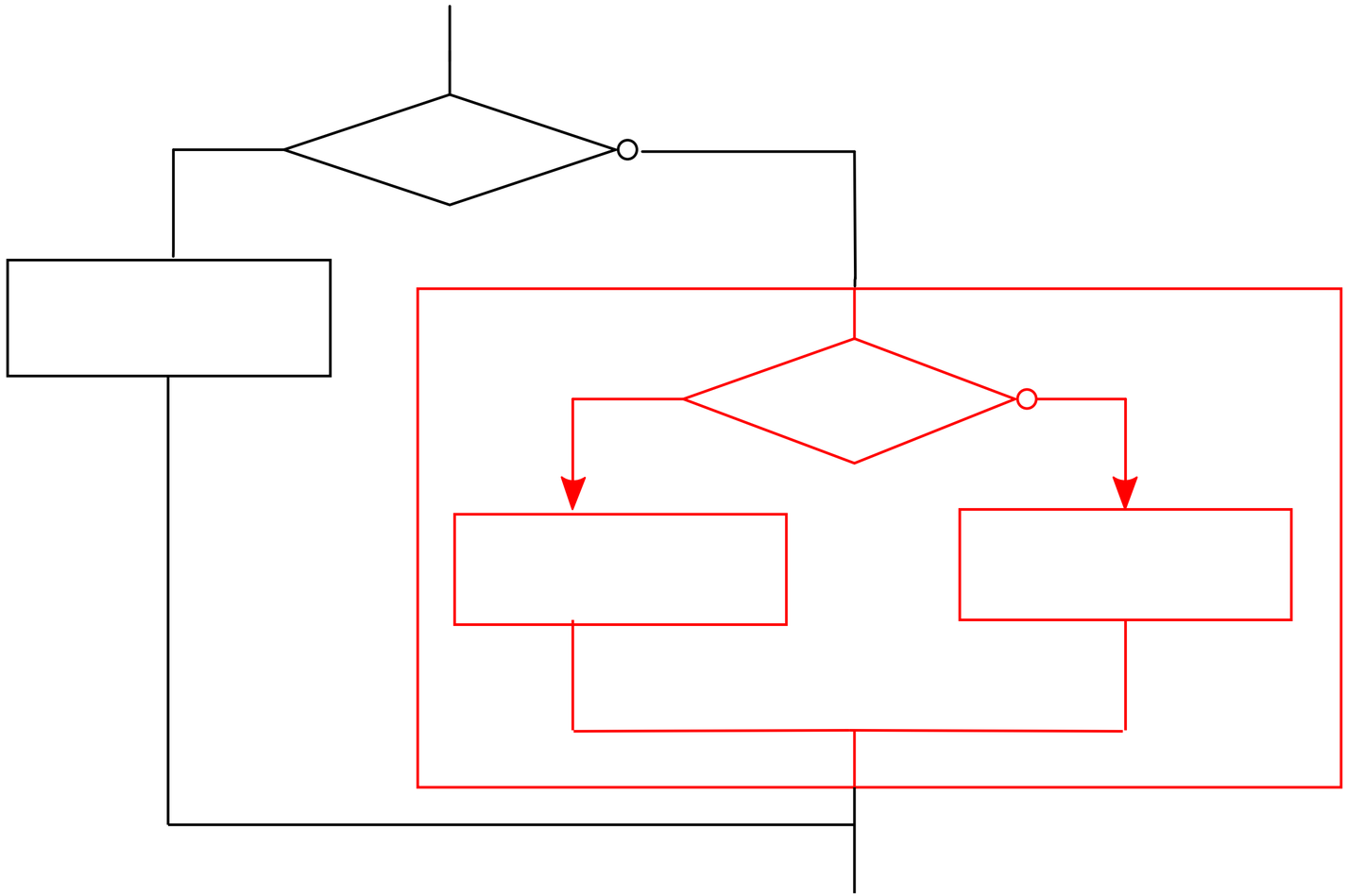
Avec le `if...else` , la condition fausse donne aussi lieu à une instruction !
Exactement `1` instruction s'exécute, soit `Instruction1`, soit `Instruction2`.

# if...else : un exemple

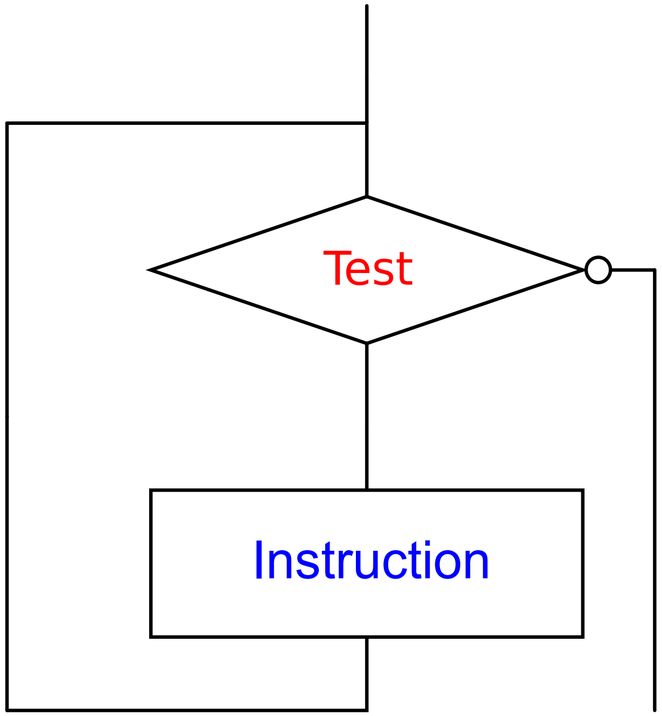
```
if (x == 0) {
   // ...
}
else if (x == 1) {
   // ...
} else {
   ///...
}
```

# if...else : un exemple

```
if (x == 0) {
   // ...
}
else if (x == 1) {
   // ...
}
else {
   ///...
}
```



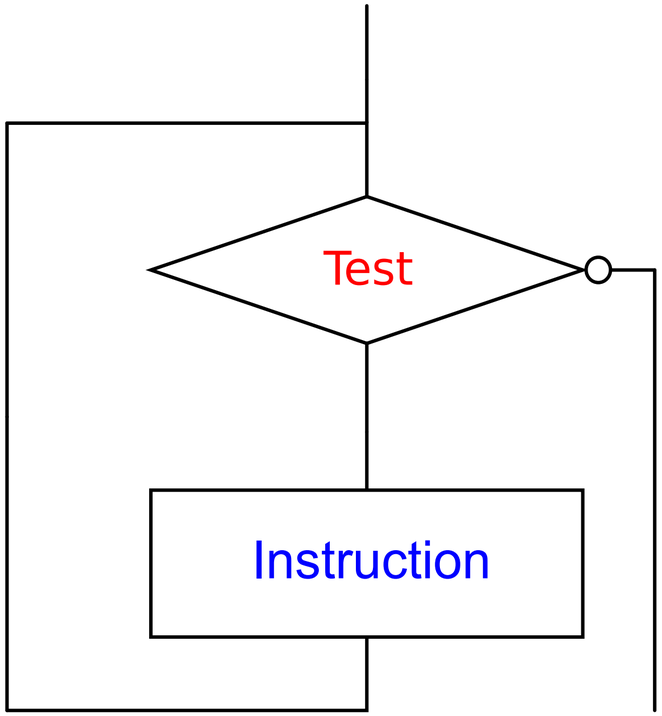
# while : la boucle avec test initial




```
while (Condition)
{
   Instruction;
}
```

```
while (Condition){
   Instruction;
}
```

# while : la boucle avec test initial

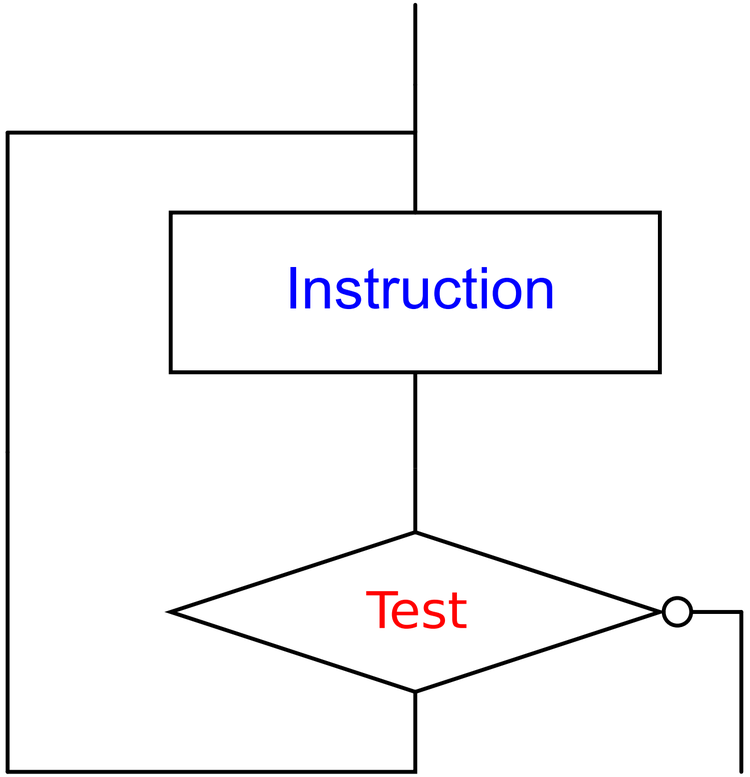



```
while (Condition)
{
   Instruction;
}
```

```
while (Condition){
   Instruction;
}
```

La structure `while` est une boucle. Elle peut durer indéfiniment !
L'instruction peut s'exécuter 0, 1 ou plusieurs fois.

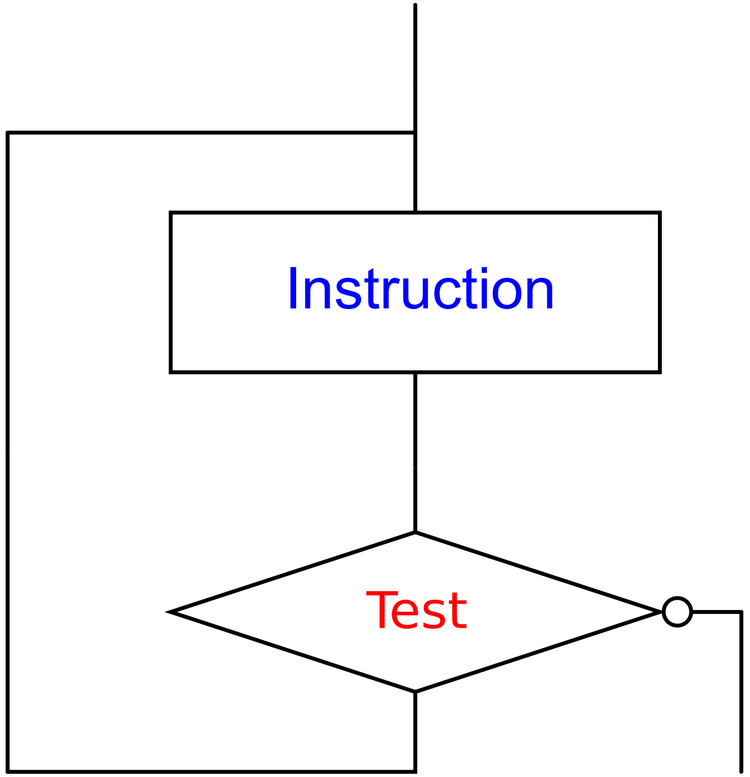
# while : la boucle avec test final




```
do
{
   Instruction;
} while (Condition);
```

```
do {
   Instruction;
} while (Condition);
```

# while : la boucle avec test final



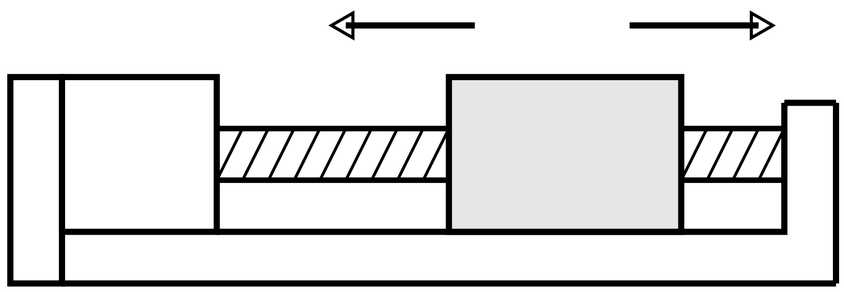

```
do
{
   Instruction;
} while (Condition);
```

```
do {
   Instruction;
} while (Condition);
```

La structure `do...while` est une boucle. Elle peut durer indéfiniment !
L'instruction peut s'exécuter 1 ou plusieurs fois.

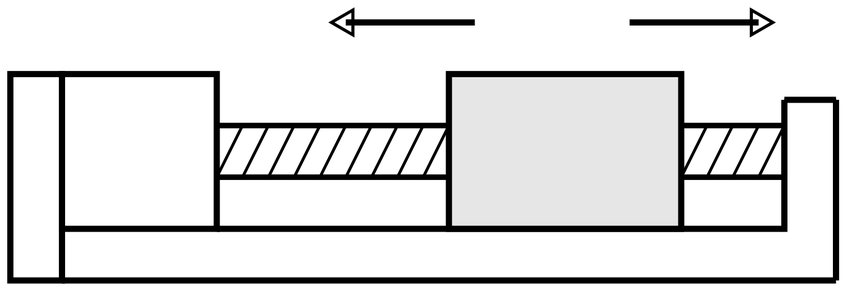
# Un exemple

```
Avance;
while (1) {
   if (FinCourseDroite){
      Avance;
   if (FinCourseGauche) {
      Recule;
   }
}
```



# Un exemple

```
Avance;
while (1) {
   if (FinCourseDroite){
      Avance;
   if (FinCourseGauche) {
      Recule;
   }
}
```



## Quizz

Quel mouvement du chariot effectue cette partie de programme ?

- Le chariot avance, puis recule et s'arrête.
- Le chariot fait un mouvement de va-et-vient permanent
- Le chariot avance, puis s'arrête

# D'autres présentations

```
while (1) {
   if (BoutonOn){
      Allume;
   }
   if (BoutonOff) {
      Eteint;
   }
}
```

```
while (1)
{
   if (BoutonOn)
   {
      Allume ;
   }
   if (BoutonOff)
   {
      Eteint;
   }
}
```

```
while (1){if(BoutonOn){Allume;} if(BoutonOff){Eteint;}}
```

# for : une autre manière d'écrire une boucle

- D'abord un exemple :

```
for (i=0 ; i<10 ; i++) {
    // instructions
}
```

- Trois paramètres :
  - **l'initialisation** : une instruction qui s'exécute une fois au début de la boucle
  - **la condition** : la condition pour que la boucle continue
  - « **l'incrémentation** » : une instruction qui s'exécute à chaque itération

# Un exemple

```
int i;
for (i=0, i<=5 ; i++) {
   AllumeLed;
   Delay (500);
   EteintLed;
   Delay 500;
}
```

# Quizz

Combien de fois la LED va s'allumer par ce programme ?

- 4 fois
- 5 fois
- 6 fois

# break : pour sortir d'une boucle

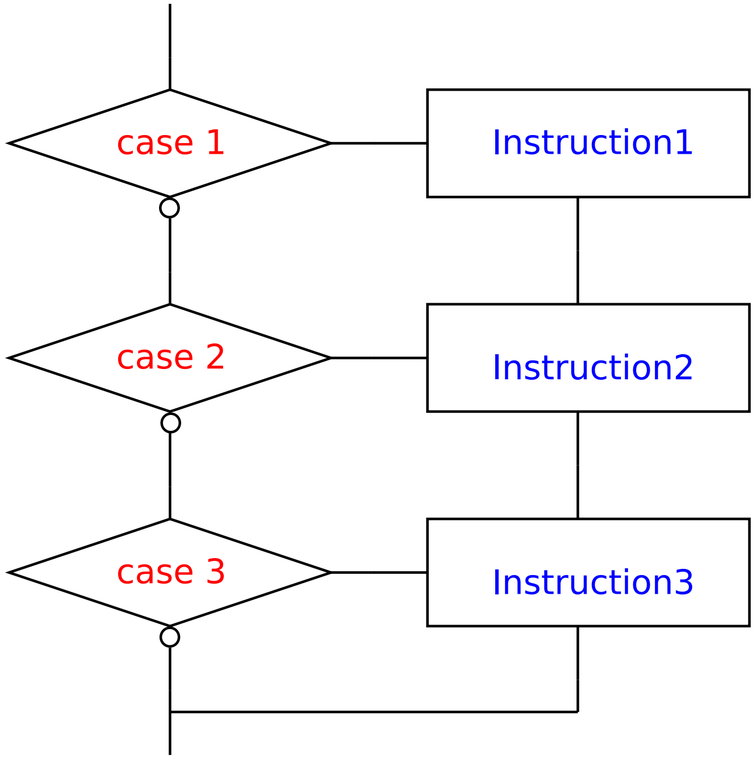
L'instruction `break` termine l'exécution de la boucle **englobante** dans laquelle elle apparaît.

# break : pour sortir d'une boucle

L'instruction `break` termine l'exécution de la boucle englobante dans laquelle elle apparaît.

```
...
while (1) {
   ...
   if (condition) {
     Break ;
   }
   ...
}
// suite du programme...
```

# switch...case : un branchement conditionnel




```
switch (expression) {
  case (1) : Instruction1;
  case (2) : Instruction2;
  case (3) : Instruction3;
}
```

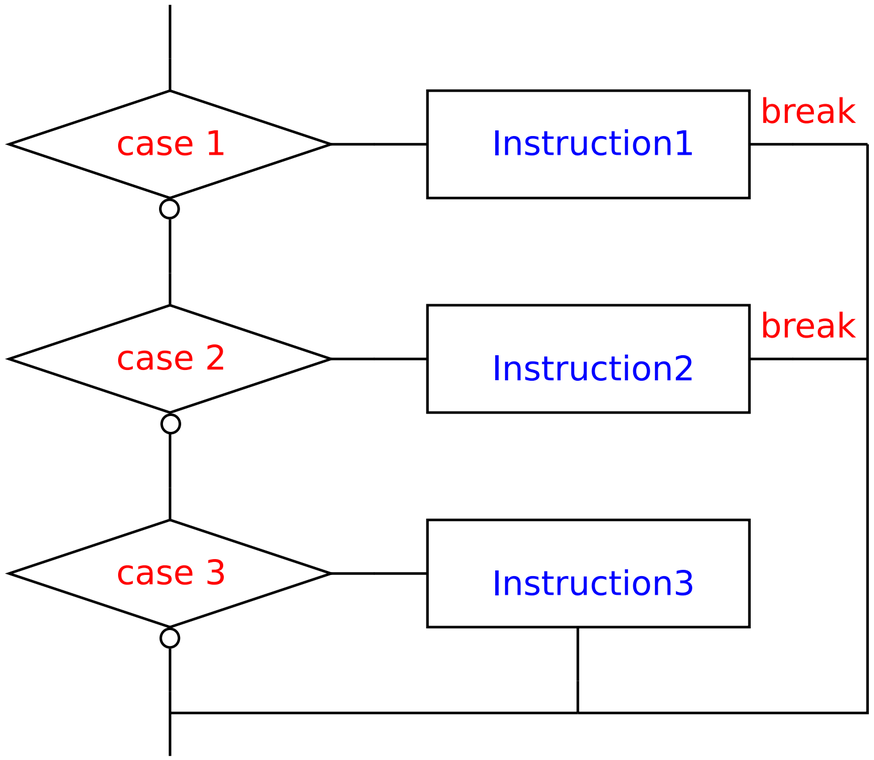
La structure `switch...case` permet d'effectuer des tests et débranchements selon une liste de valeurs d'une variable.

# switch...case : un branchement conditionnel

```
switch (expression) {
  case (1) : Instruction1;
  case (2) : Instruction2;
  case (3) : Instruction3;
}
```

```
switch (expression) {
  case (1) : Instruction1;
  case (2) : Instruction2;
  case (3) : Instruction3;
  default : InstructionDef;
}
```

# switch...case : un branchement conditionnel




```
switch (expression) {
  case (1) : Instruction1;
             break;
  case (2) : Instruction2;
             break;
  case (3) : Instruction3;
             // break;
}
```

Dans presque tout les cas, la structure `switch...case` s'utilise avec des `break`

*Attention de ne pas les oublier !*

# Procédures

- Exemples de **procédure**

```
void Clign (int nombre) {
  for (i=0; i<nombre; i++) {
    Allume; Delay(500); Eteint; Delay(500);
  }
}
```

```
int Carre (int nombre) {
  return nombre*nombre;
}
```

# La procédure main : le programme principal

- Une procédure particulière : **main**

```
void main () {
  // instructions...
}
```

```
int main () {
  setup() ;
  while (1) {
    loop() ;
  }
}
```

# Comprendre les Microcontrôleurs

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

- Variables et assignations
- Structures de contrôle
- Procédures
  et programme principal
- Ce ne sont pas tous les éléments
  du langage C
  *On en verra d'autres plus tard.*