

Cours de langage C

Mireille Goud – HEG ESVIG

mireille.goud@hegvd.ch



Objectifs du cours

- Apprentissage d'un langage de programmation.
- Connaître les fonctionnalités du langage C.
- Savoir coder un algorithme en langage C et le faire fonctionner.



Programme

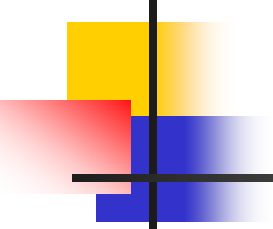
- Chapitre 1 : Introduction
- Chapitre 2 : Les variables
- Annexes : Fonctions printf et scanf
- Chapitre 3 : Expressions et opérateurs
- Chapitre 4 : Instructions de contrôle
- Chapitre 5 : Les tableaux
- Chapitre 6 : L'utilisation de pointeurs



Programme

- Chapitre 7 : Chaînes de caractères
- Chapitre 8 : Fonctions de bibliothèques
- Chapitre 9 : Tableau de pointeurs
- Chapitre 10 : Allocation dynamique de mémoire
- Chapitre 11 : Les Fonctions
- Chapitre 12 : Les structures
- Chapitre 13 : Les fichiers

Historique

- 
- 1951 : A0 – Premier langage évolué et compilé
 - 1954 : Fortran (John Backus – IBM – langage propriétaire)
 - 1958 : ALGOL
 - 1960 : Cobol
 - 1964 : Basic
 - 1970 : Pascal
 - 1970 : B (Thompson, Bell Lab)
 - 1972 : Langage D crée par Dennis Ritchie (Bell Telephon Lab)
 - 1973 : UNIX traduit en C sur PDP-11
 - 1975 : Ada
 - 1983 : C++ (Stroustrup, Bell Lab)
 - 1995 : Java (SUN)



Utilisation

- système d'exploitation (Unix, Linux)
- traitement de texte
- Langage C++ : version améliorée du C (Orienté objet)



Caractéristiques

- Langage de haut niveau
- Jeu d'instruction varié
 - Opération arithmétique
 - Logiques
 - Manipulation de bits
- Récursivité



Caractéristiques

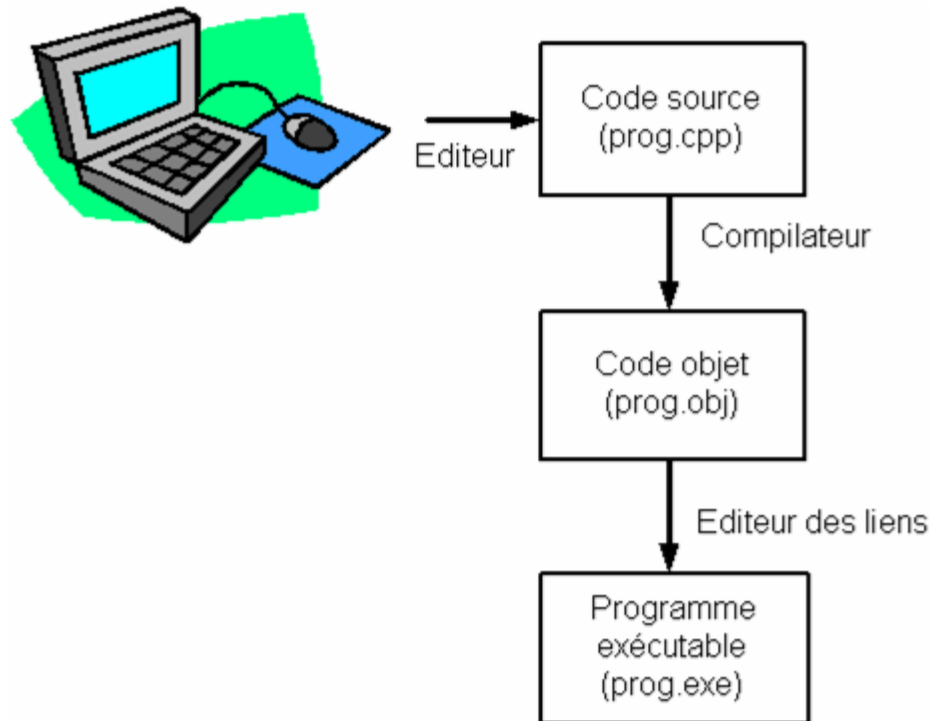
- Modulaire
 - Définition de fonctions
 - Compilation par modules (Une application peut être développée avec plusieurs fichiers)
- Portable
- Simplicité

Développement d'un programme



- 1. Définir les objectifs du programme
- 2. Ecrire le ou les algorithmes de traitement
- 3. Ecrire le programme dans le langage à l'aide d'un éditeur
- 4. Compiler le programme
- 5. Exécuter le programme
- 6. Mise au point du programme tant que l'exécution ne correspond pas aux objectifs. (Corriger le programme à l'aide de l'éditeur, compiler le programme corrigé et exécuter le programme soit retour aux points 3, 4, 5)

Création d'un fichier exécutable





Programme

```
#include <stdio.h>
```

```
void main () {
```

```
    /*Imprimer un message */
```

```
    printf ("Salut, programmeur !\n");
```

```
}
```



Les variables

- La mémoire : chaque octet a une adresse mémoire.
- Un identificateur :
 - Nom des variables, constantes, fonctions.
 - Commence par une lettre ou '_'
 - Ne doit pas être un mot réservé.



Mots réservés

asm	const	else	goto	return	struct	void
auto	continue	enum	if	short	switch	volatile
break	default	extern	int	signed	typedef	while
case	do	float	long	sizeof	union	goto
char	double	for	register	static	unsigned	return



Les variables

- Déclaration d'une variable
 - Nom : Identificateur
 - Type : définit la représentation en mémoire et les opérations applicables.

char	caractère
int, short, long	entier
float, double	Nombre à virgule flottante

Unsigned pour les caractères et entiers non signés



Déclaration de variables

- type nom ;

`int` longueur, largeur

`float` note, moyenne

`char` c ;

Type : dépend de la nature de l'information traitée.

Nom : donner un nom significatif.



Type de variable

<i>Mot clé</i>	<i>Type</i>	<i>Nb octets</i>	<i>Valeurs</i>
char	caractères	1	'a', 'A' '?'
int	entier	4	-2 147 483 648 à 2 147 483 647
long	entier long	4	Idem int
short	entier court	2	-32 768 à 32767



Type de variables (2)

<i>Mot clé</i>	<i>Type</i>	<i>Nb octets</i>	<i>Valeurs</i>
float	Nombre réel	4	$1,2 \times 10^{-38}$ à $3,4 \times 10^{38}$
double	Nombre réel Double précision	8	$2,2 \times 10^{-308}$ à $1,8 \times 10^{308}$
unsigned int	Nombre positif	4	0 à 4 294 967 295



Structure de la fonction main

```
void main () {
```

Déclaration de toutes les variables

Instructions

```
}
```



Initialisation d'une variable

- Opérateur d'affectation : '='

```
int x;
```

```
x = 12;  /* Donne la valeur 12 à la variable x */
```

La valeur doit être du même type que la variable.

```
int x = 12; /* Initialisation à la déclaration */
```



Ecriture d'une variable

- Ecriture : fonction printf

%d	int
%c	char
%f	float

```
int valeur1;
```

```
float valeur2;
```

```
printf ( "valeur entière : %d, valeur réelle : %f \n",  
        valeur1 * 2, valeur2 );
```



Printf : exemple

```
int jour = 4 ;
```

```
int mois = 8 ;
```

```
int annee = 1996 ;
```

```
printf ("Date de naissance : %d %d %d\n",  
        jour, mois, annee);
```

Donnera le résultat suivant :

```
Date de naissance : 4 8 1996
```



Lecture d'une valeur au clavier

- fonction scanf

```
scanf( "Un ou plusieurs formats" ,  
      &variable1, &variable2, ...);
```

- mêmes formats % que printf
- un format par valeur lue au clavier
- opérateur '**&**' devant le nom de la variable pour donner l'adresse de la variable.



Example : scanf

```
float note ;
```

```
printf ("Note : ");
```

```
/* Initialisation de note par la valeur  
   saisie au clavier */
```

```
scanf ("%f", &note);
```

ATTENTION : Le message entre "" ne contient que des formats et pas d'autres caractères.



Instruction

- Tâche à accomplir par le processeur :
 - Mettre une valeur dans une variable
 - Faire une opération arithmétique
 - Imprimer un message
 - Lire une valeur au clavier pour l'affecter à une variable
 -



Expression

- Évaluation d'une expression :

`x = 8 + 34;`

L'opération `8 + 34` est effectuée et le résultat est affecté à la variable `x`.



Opérateurs

- Opération ou action à effectuer sur un ou plusieurs opérandes.
 - Opérateur d'affectation : =
 - Opérateurs mathématiques : + - * / %
 - Opérateurs de comparaison : < > >= <= == !=
 - Opérateurs logiques : ! && ||
 - Opérateurs binaires : ~ & | ^ >> <<



Opérateurs arithmétiques

+	Addition	$x = x + 10;$
-	Soustraction	$x = x - y;$
*	Multiplication	$x = y * 5;$
/	Division	$x = x / 2;$
%	Modulo	$x = y \% 2;$
++	Incrémentation	$x++;$
--	Décrémentation	$x = y--;$



Opérateurs et types

Opérateurs	Types
<code>-- + * / %</code> <code>--++ --</code>	int long short char (division entière)
<code>- + * /</code>	float double



Hiérarchie des opérateurs

- Règles de priorité pour appliquer les opérateurs

Opérateur	Priorité
++ --	1
* / %	2
+ -	3

Si égalité : évaluation de gauche à droite



Opérations arithmétiques et types

- L'opérateur % (modulo) ne s'applique pas aux float.
- Lorsque les 2 opérandes d'un opérateur binaire sont de types différents, ils sont convertis en un type commun.
L'opérande du type le plus « étroit » est converti dans le type de l'opérande du type le plus « large »



Opérations arithmétiques et types

- L'opérateur % : int, long, short, char.
- 2 opérandes de types différents

=> conversion en un type commun.

L'opérande du type le plus « étroit » est converti dans le type de l'opérande du type le plus « large »



Conversion de types

- Exemple :

- $\text{int} + \text{float} \Rightarrow \text{résultat float}$

- l'opérande de type `int` est converti en `float` \Rightarrow le résultat sera du type `float`.

- $\text{float} + \text{double} \Rightarrow \text{résultat double}$

- l'opérande de type `float` est converti en `double` \Rightarrow le résultat sera du type `double`.



Affectation et types

- float = double => conversion du double en float avec un *warning*
- short = int => conversion du int en short avec *warning*
- ATTENTION : il y a un risque de tronquer une valeur



Opérateurs d'affectation

Symbole	Exemple	Equivalent à
+=	x += 2;	x = x + 2;
-=	x -= y;	x = x - y;
*=	x *= x;	x = x * x;
/=	x /= y - 1;	x = x / (y - 1);
%=	x %= 2;	x = x % 2;



Opérateurs relationnels

- Comparaisons d'expressions
résultat : 0 si faux et 1 si vrai

<code>==</code>	Egal : <code>x == 20</code>
<code>></code>	Supérieur : <code>x > y</code>
<code><</code>	Inférieur : <code>x + y < 100</code>
<code>>=</code>	Supérieur ou égal : <code>x >= 0</code>
<code><=</code>	Inférieur ou égal : <code>x <= 0</code>
<code>!=</code>	Différent : <code>x != 0</code>



Valeur logique

- Pas de valeur booléenne en langage C
- On utilise une valeur numérique (short, int)

FAUX	0
VRAI	$\neq 0$



Opérateurs logiques



!	NON	1
&&	ET	2
	OU	3



Exemple opérateurs logiques

- `int a, b, trouve ;`
- `(a >= 0) && (b <= 0)`
- `(a == 0) || (b == 0) || ! trouve`



Opérateurs binaires

~	complément à 1 (opérateur unaire)
&	et
	ou inclusif
^	ou exclusif
>>	décalage à droite
<<	décalage à gauche

Attention : Ne pas confondre avec les opérateurs relationnels



Ordre d'évaluation des opérateurs

- A connaître pour les expressions simples :
 - $-x * 2 + 3 \Rightarrow (-x * 2) + 3$
- Utiliser des parenthèses pour les expressions complexes :
 - $(x * y) + (z / (x + 3))$



Exécution d'un programme

- Début : 1ère instruction de la fonction main.
- Fin : dernière instruction de la fonction main.
- Sans instruction de contrôle, toutes les instructions sont exécutées séquentiellement.



Instruction if

```
if (expression) {  
    instructions;  
}
```

```
if (note == 6) {  
    printf(" BRAVO"); //note == 6  
}
```



La clause else

```
if (condition) {  
    instructions1; // expression  
                  // vraie  
}  
else {  
    instructions2; // expression  
                  // fausse  
}
```



Exemple

```
if (age < 16) {  
    printf ("Mineur"); // age < 16  
} else {  
    printf ("Majeur"); // age >= 16  
}
```



if imbriqués

```
if (condition1) {  
    instructions1; //condition1 Vraie  
                  //condition2 ?  
  
} else if (condition2) {  
    instructions2; //condition1 Faux  
                  //condition2 Vraie  
  
} else {  
    instructions3; //condition1 Faux  
                  //condition2 Faux  
  
}
```



Exemple

```
if (age < 16) {  
    printf ("Junior"); //age < 16  
} else if (age < 65) {  
    printf ("Major"); //age >= 16  
                        // et age < 65  
} else {  
    printf ("Senior"); //age >= 65  
}
```



Instruction switch

- Instruction à choix multiple
- Remplace une instruction if imbriquée
- Attention : la valeur testée doit être une valeur entière (short, int, long, char).
- Valeurs proposées (case) : constantes.



Instruction Switch

```
switch (expression) {  
    case constante1 :  
        Instructions1 ;  
        break ;  
    case constante2 :  
        Instructions2 ;  
        break ;  
    case constanteN :  
        instructionsN ;  
        break ;  
    default ://expression différente des constantes  
        instructionsD ;  
        break ;  
}
```




Exemple : instruction switch

```
#include <stdio.h>
void main () {
    int i,j;
    char c;
    printf ("Entrer 'o' pour oui et 'n' pour non : ");
    scanf ("%c", &c);
    switch (c) {
        case 'o':
            printf ("Oui\n");
            break;
        case 'n':
            printf ("Non\n");
            break;
        default :
            printf ("Vous devez entrer 'o' ou 'n'\n");
    }
}
```



Exemple 2 - switch

```
#include <stdio.h>
void main () {
    int i,j;
    char c;
    printf ("Entrer 'o' ou 'y' pour oui et 'n' pour non : ");
    scanf ("%c", &c);
    switch (c) {
        case 'o':
        case 'y':
            printf ("Oui\n");
            break;
        case 'n':
            printf ("Non\n");
            break;
        default :
            printf ("Vous devez entrer 'o' ou 'n'\n");
    }
}
```

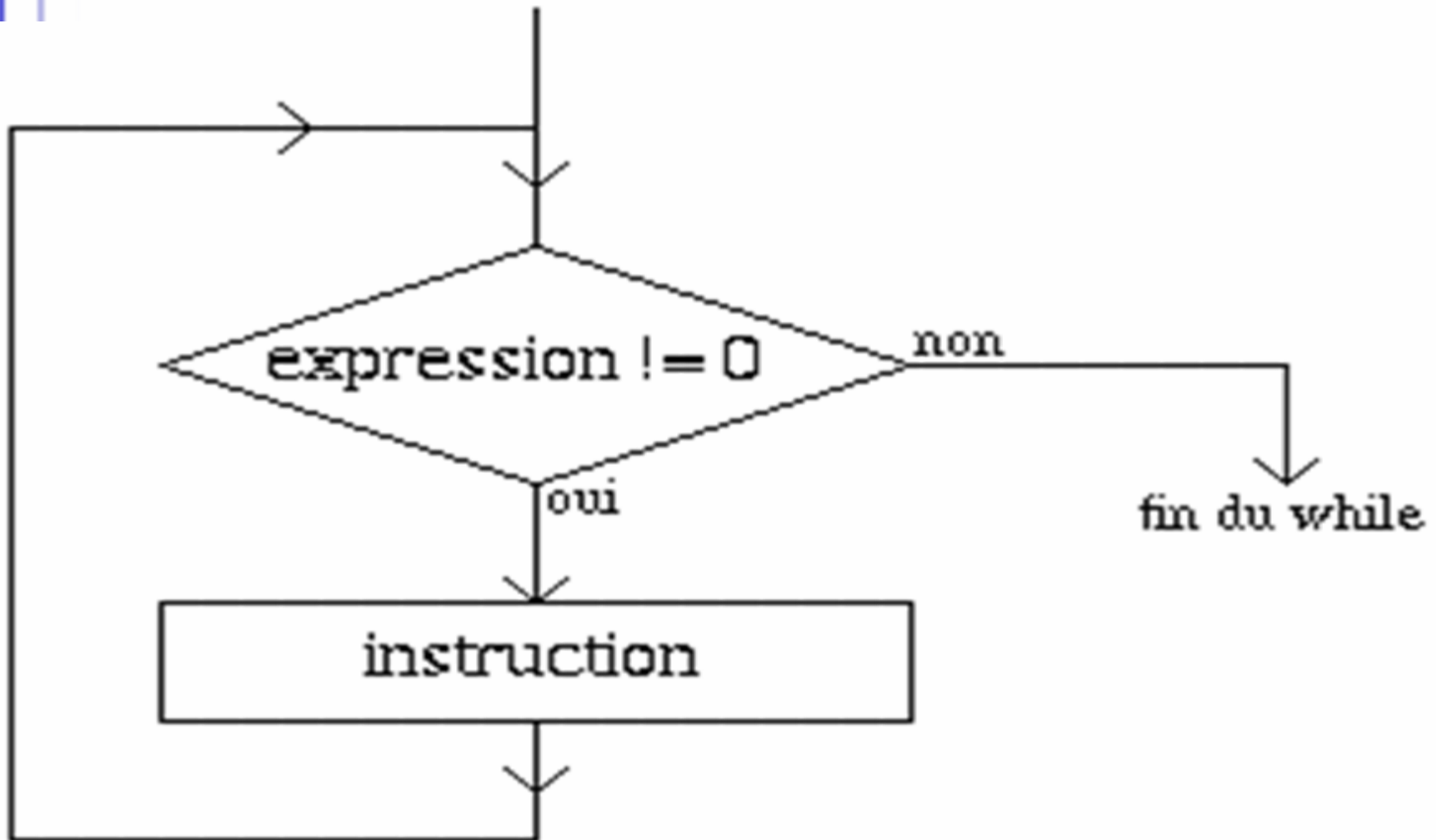


Boucle while

```
while ( expression ) {  
    instructions;  
}
```

- les *instructions* sont exécutées tant que le résultat de *expression* est vrai. (\neq de 0).
- *expression* est évaluée avant chaque exécution des *instructions*

Instruction while





Boucle while : Exemple 1

```
int i;

/* affiche les nombres de 0 - 9 */
i = 0; // Initialisation de la boucle
while (i != 10) { //Test condition
    printf("%d ", i);
    i++; // Passage à la valeur suivante
}
```



Boucle while : Exemple 1

```
int n, val = 1;
int somme = 0;
/* Lire la première valeur */
printf("Entrer une valeur entiere : ");
n = scanf("%d", &val);
while (val != 0) {
    // Traitement de la valeur
    if (n != 1) {
        puts ("Erreur de saisie");
        fflush (stdin); // Vider le buffer de saisie
    } else {
        somme += val;
    }
    /* Lire la valeur suivante */
    printf("Entrer une valeur entiere : ");
    n = scanf("%d", &val);
} //Fin while
printf ("La somme est : %d\n", somme);
```

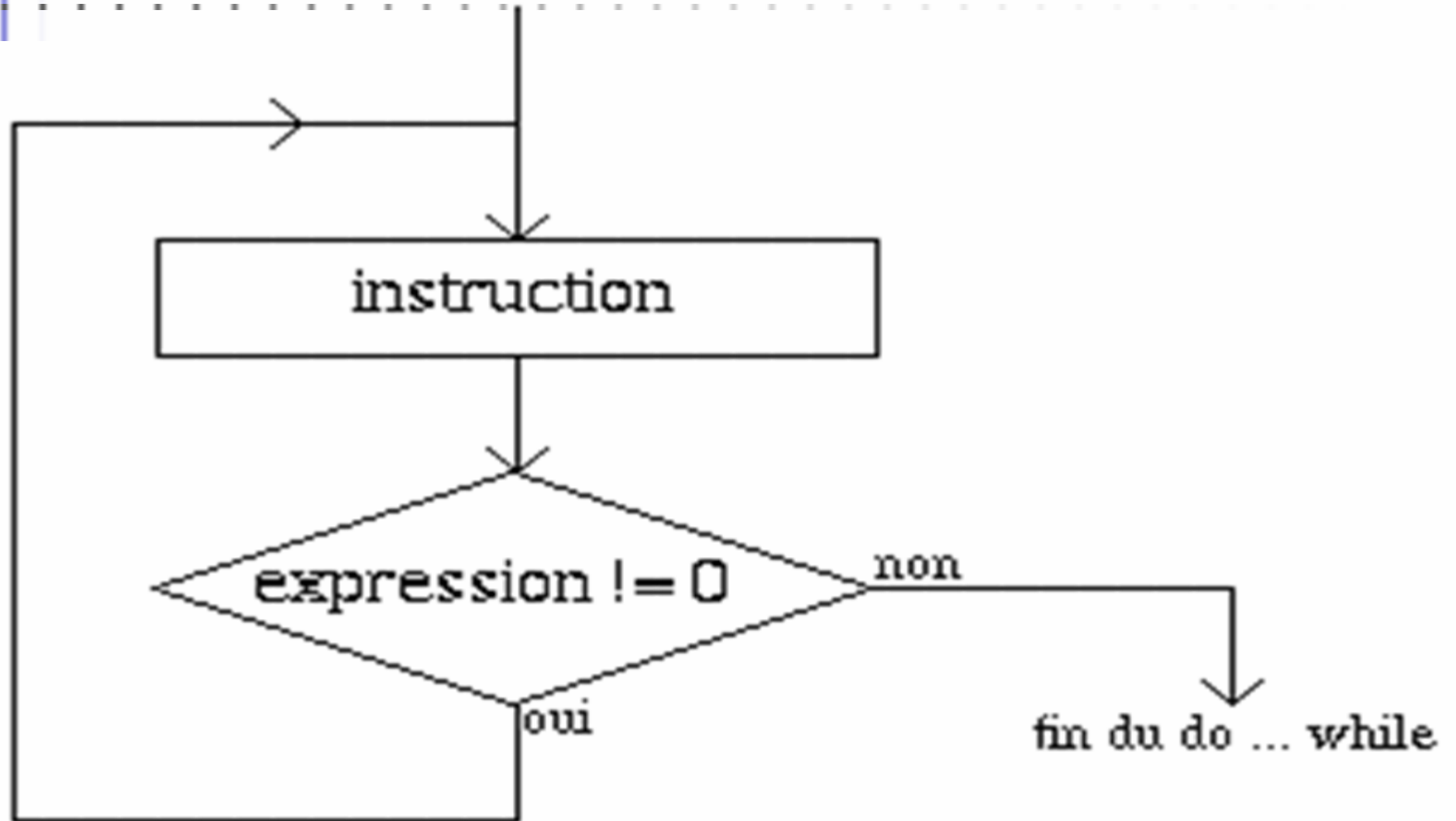


Boucle do while

```
do {  
    instructions;  
} while ( expression != 0 );
```

- Cette instruction est similaire à la boucle while mais le test a lieu après chaque exécution des *instructions*, donc les *instructions* sont exécutées au moins une fois.

Boucle do while





Boucle do while : exemple

```
int valeur ;  
/* Saisie d'une valeur tant que la valeur n'est pas  
   comprise entre 0 et 20 */  
do {  
    printf("Entrer une valeur entre 0 et 20 : ");  
    n = scanf ("%d", &valeur);  
    fflush (stdin);  
} while (n !=1 || valeur < 0 || valeur > 20);
```



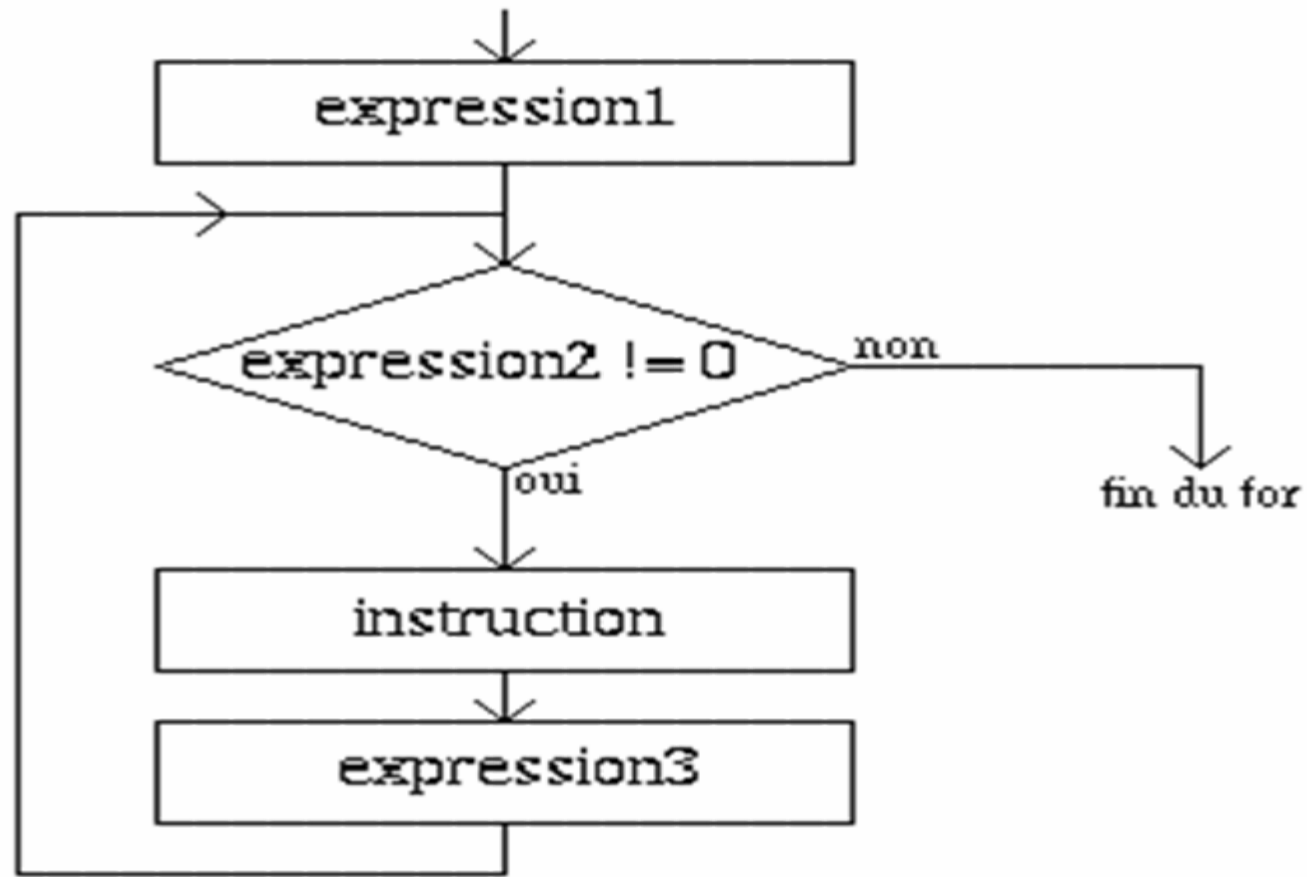
Boucle For

```
for( expression1 ; expression2 ; expression3 ) {  
    instructions  
}
```

- La boucle for est équivalente à la structure suivante:

```
expression1;  
while ( expression2 ) {  
    instructions  
    expression3;  
}
```

Diagramme Boucle For





Exemple Boucle For

```
int i ;  
/* affiche les nombres de 0 - 9 */  
for (i = 0; i < 10; i++) {  
    printf("%d ", i);  
}
```

Résultat :

0 1 2 3 4 5 6 7 8 9