

Mise à niveau Langage C



Licence SPI3

**Christophe Després
Maître de Conférences**

Deuxième partie



Les types fondamentaux du C

Types fondamentaux du C

■ *Les entiers*

- *Les caractères* *char*
- *Les entiers courts* *short int*
- *Les entiers long* *long int*
- *Les entiers classiques* *int*

■ *Les réels*

- *Les réels simple précision* *float*
- *Les réels double précision* *double*
- *Les réels quadruple précision* *long double*

Les caractères

- *En C un caractère est un entier car il s'identifie à son code ASCII (le plus souvent).*
- *Un caractère peut donc être codé sur un octet.*
- *On peut donc appliquer toutes les opérations entières sur les caractères (addition, soustraction, etc.).*

c = 'a' => c = 97

c = c + 1

c = 'b' => c = 98

Les entiers



- *Un entier correspond généralement à un mot machine*
- *Les attribut short, long, unsigned peuvent qualifier un entier*
 - *long int x; /* x est un entier long (≥ 32 bits) */*
 - *short int x; /* x est un entier court */*
 - *unsigned int x; /* x est un entier non signé */*
 - *unsigned short int x; /* x est un entier court non signé */*

Les entiers

- *Il n'y a pas de taille fixée par le langage, mais le C garanti que :*
 - $1 = T_C \leq T_S \leq T_I \leq T_L$
- *Lorsque l'on utilise les attributs short, long ou unsigned, on peut omettre le nom du type int*
 - *long x;*
 - *short x;*
 - *unsigned x;*
 - *unsigned short x;*

Les réels



- *Les float sont des réels codés de manière interne sous forme de mantisse/exposant*
- *Les double sont des float plus long et les long double sont des doubles plus long*
- *Plus la taille est grande plus la mantisse est grande et plus la précision est grande*
- *Comme pour les entiers ces tailles sont dépendantes de la machine*

Les réels



- *Le C garanti juste que*

- $T_F \leq T_D \leq T_{LD}$

- *Tailles minimales*

- *char* 1 octet

- *short int* 2 octets

- *int* 2 octets

- *long int* 4 octets

- *float* 4 octets

- *double* 8 octets

Les constantes



- *Constantes de type entier*
- *Constantes de type char*
- *Constantes de type logique*
- *Constantes de type réel*
- *Constantes de type chaîne de caractères*

Constantes de type entier

- *On dispose de 3 notations pour les constantes entières*
 - *décimale, qui ne commence pas par un 0, ex : 234*
 - *octale (base 8), qui commence par un 0 suivi d'un chiffre, ex : 0234 (=156 en base 10)*
 - *héxadécimale (base 16), qui commence par un 0 suivi d'un x ou d'un X, ex : 0xC (=12 en base 10)*

Constantes de type entier

- Une constante numérique en base 10 est normalement un `int`. Si le nombre est trop grand le compilateur essaiera dans l'ordre les types `long int`, `unsigned long int`
- Une constante numérique en base 8 ou 16 est normalement un `int`. Si le nombre est trop grand le compilateur essaiera dans l'ordre les types `unsigned int`, `long int`, `unsigned long int`
- Les constantes de type `short int` et `unsigned short int` n'existent pas

Constantes de type char

- *Elles n'existent pas réellement en C*
- *Une constante caractère est de type int et a pour valeur le code du caractère dans le codage utilisé par la machine*
- *Une constante caractère s'écrit entourée du signe '*
- *La constante caractère correspondant au caractère a s'écrit 'a'*
- *+ s'écrit '+'*
- *1 s'écrit '1'*

Constantes de type char

■ *Caractères non-imprimable*

sémantique	caractère
newline	' \n '
horizontal tabulation	' \t '
vertical tabulation	' \v '
back space	' \b '
carriage return	' \r '
audible alert	' \a '

Constantes de type char

■ *Caractères spéciaux*

sémantique	caractère
'	' \ ' '
"	' \ " '
\	' \\ '

Constantes de type logique

- *Les constante de type logique n'existent pas en tant que telle en C. On utilise la convention suivante sur les entiers*
 - *$0 \Leftrightarrow$ faux et tout le reste est vrai*

Constantes de type réel

■ *Les constantes réelles sont de la forme*
<partie entière>.<partie fractionnaire><e ou
E><partie exposant>

■ *On peut omettre*

➤ *soit <partie entière> : .475e-12*

➤ *soit <partie fractionnaire> : 482.*

➤ *Mais pas les deux*

■ *On peut omettre*

➤ *soit . : 12e7*

➤ *soit <partie exposant> : 12.489*

➤ *mais pas les deux*

Constantes de type réel

- *Une constante réelle non suffixée a le type double*
- *Une constante réelle suffixée par *f* ou *F* a le type float*
- *Une constante suffixée par *l* ou *L* a le type long double*

Constantes de type chaîne de caractères

- *Elles sont constituées de caractères quelconques encadrés par des guillemets, ex : "abc"*
- *Elle sont stockées en mémoire statique sous la forme d'un tableau de caractères terminé par la valeur '\0' (valeur entière 0)*

a	b	c	\0
---	---	---	----

Les constantes nommées

- *Il y a trois façons de donner un nom à une constante : soit en utilisant les possibilités du préprocesseur, soit en utilisant des énumérations, soit avec le mot-clé const*
 - *#define*
 - *enum*
 - *const (depuis la norme ANSI)*

#define



- *#define identificateur reste-de-la-ligne*
- *Le préprocesseur lit cette ligne et remplace dans toute la suite du source, toute nouvelle occurrence de identificateur par reste-de-la-ligne*
- *#define PI 3.14159*
 - *et dans la suite du programme on pourra utiliser le nom PI pour désigner la constante 3.14159*
- *Il s'agit d'une transformation d'ordre purement textuel*

#define

- *Une telle définition de constante n'est pas une déclaration mais une commande du préprocesseur. Il n'y a donc pas de ; à la fin*
- *Si on écrit :*
`#define PI 3.14159;`
 - *le préprocesseur remplacera toute utilisation de PI par 3.14159; et par exemple remplacera l'expression PI / 2 par 3.14159; / 2 ce qui est une expression incorrecte.*
 - *Dans une telle situation, le message d'erreur ne sera pas émis sur la ligne fautive (le #define), mais sur une ligne correcte (celle qui contient l'expression PI / 2), ce qui gênera la détection de l'erreur.*

Les énumérations

- *On peut définir des constantes de la manière suivante :*

enum { liste-d'identificateurs }

- *enum {LUNDI, MARDI, MERCREDI, JEUDI};*

➤ *définit les identificateurs LUNDI,..., JEUDI comme étant des constantes de type int, et leur donne les valeurs 0, 1, 2, 3.*

- *enum {FRANCE = 10, ESPAGNE = 20};*

- *enum {FRANCE = 10, ITALIE, ESPAGNE = 20};*

➤ *ITALIE à la valeur 11*

- *enum {ESPAGNE = 'E', FRANCE}*

Déclaration des variables

- *Les noms de variables (et de constantes) sont constitués de lettres et de chiffres*
 - *le premier caractère doit être une lettre*
 - *le caractère souligné noté "_" est considéré comme une lettre*
 - *les caractères majuscules et minuscules sont différents, en langage C on utilise généralement les majuscules pour les constantes et les minuscules pour les variables*
- *Une déclaration indique un certain type et regroupe derrière une ou plusieurs variables*
 - *<Type> <NomVar1>, <NomVar2>, ..., <NomVarN>;*
- *Les variables doivent être déclarées avant d'être utilisées*

Déclaration des variables



■ **Exemple 1 :**

- *int compteur,X,Y;*
- *float hauteur,largeur;*
- *double masse_atomique;*

■ **Exemple 2 :**

- *int compteur*
- *int X;*
- *int Y;*
- *float hauteur;*
- *float largeur;*
- *double masse_atomique;*

Initialisation des variables

- *L'initialisation des variables peut se faire au moment de la déclaration*
- *Exemples :*
 - *int max = 1023;*
 - *char tab = '\t';*
 - *float x = 1.05e-4;*

Troisième partie



Expressions & opérateurs

Expressions



- *Une expression est une notation de valeur*
- *L'évaluation d'une expression est le processus par lequel le programme obtient la valeur désignée.*
- *Les constantes et les variables sont des expressions (elles retournent une valeur)*
- *Une expression (complexe) est constitué d'opérandes et d'opérateurs*
- *Les opérandes dénotent les valeurs à composer. Ce sont elles-mêmes des expressions (qui peuvent être des constantes ou des variables)*

Les opérateurs



- *Opérateur d'incrémentatation et de décrémentation*
- *Opérateurs arithmétiques*
- *Opérateurs relationnels*
- *Opérateurs logiques*
- *Opérateurs binaires*
- *Opérateur d'affectation*
- *Opérateur conditionnel*
- *Opérateur de cast*

Opérateur d'in(dé)crémentation

- *L'opérateur ++ ajoute 1 à son opérande*
- *L'opérateur -- retranche 1 à son opérande*
- *Lorsque l'opérateur est placé devant l'opérande, l'in(dé)crémentation a lieu avant l'utilisation de la valeur de l'opérande*
- *Exemples :*
 - *n = 5;*
x = ++n;
=> x vaut 6 et n vaut 6;
 - *n = 5;*
x = n--;
=> x vaut 5 et n vaut 4;

Les opérateurs arithmétiques

- + *Addition*
 - - *Soustraction (ou opérateur unaire)*
 - * *Multiplication*
 - / *Division entière et réelle*
 - % *reste de la division entière (Modulo)*
-
- *Ces opérateurs (sauf le %) sont applicables aussi bien à des entiers qu'à des réels*
 - *Dans le cas de 2 opérandes de type entier, le résultat de la division est entier, dans tous les autres cas, il est réel*

Opérateurs relationnels

- == *égalité (<> de l'affectation)*
- != *différence*
- > *supérieur*
- >= *supérieur ou égal*
- < *inférieur*
- <= *inférieur ou égal*

Opérateurs relationnels

- *Les deux opérandes doivent avoir le même type arithmétique. Si ce n'est pas le cas, des conversions sont effectuées automatiquement*
- *Le type `BOOLEAN` n'existe pas explicitement en C : les opérateurs de relation fournissent les valeurs 0 ou 1 (respectivement `FAUX` et `VRAI`) du type `int`*
- *En C, on peut écrire $A < B < C$ car cette expression correspond à $(A < B) < C$ ce qui n'est probablement pas le résultat escompté par le programmeur. En effet si $A < B$ est vrai, l'expression équivaut à $1 > C$ et sinon à $0 < C$*

Opérateurs logiques

- `!` *Négation unaire d'une valeur logique*
- `&&` *ET de 2 valeurs logiques*
- `||` *OU de 2 valeurs logiques*
- *Ces opérateurs interviennent sur des valeurs de type int (0 => FAUX, toutes les autres => VRAI)*
- *Les valeurs produites sont 0 (FAUX) ou 1 (VRAI)*
- *L'opérande gauche est évaluée avant celle de droite pour les opérateurs `&&` et `||`*

Opérateurs logiques

- *L'opérande de droite peut ne pas être évaluée si celle de gauche suffit à déterminer le résultat*
 - *0 à gauche d'un && implique FAUX*
 - *1 à gauche d'un || implique VRAI*
- *Exemple :*
 - int tab[10];*
 - soit le test : (k<10) && (tab[k]!=v)*
- *Si k est supérieur ou égal à 10, l'expression tab[k]!=v ne sera pas évaluée et il vaut mieux car pour k supérieur ou égal à 10 tab[k] est indéterminée*

Opérateurs binaires

- *Ces opérateurs sont au ras de la machine, ils servent à manipuler des mots bit à bit pour faire des masques par exemple.*
- *$a \& b$: et binaire \Rightarrow mettre des bits à 0*
 - *$c = n \& 127$: c sera constitué des 7 bits de poids faible de n et complété à gauche par des 0*
- *$a | b$: ou binaire \Rightarrow mettre des bits à 1*
 - *$c = n | 127$: c sera constitué de 7 bits de poids faible à 1 et des bits de poids fort de n*
- *$a \wedge b$: ou exclusif binaire \Rightarrow inverser des bits*
 - *$c = n \wedge 127$: c sera constitué des bits de n avec les 7 bits de poids faible inversés*

Opérateurs binaires

■ $a \ll b$: *décalage à gauche*

- Les bits de a sont décalés de b positions vers la gauche, les bits de poids fort sont perdus, des 0 arrivent sur la droite

■ $a \gg b$: *décalage à droite*

- Les bits de a sont décalés de b positions vers la droite, les bits de poids faible sont perdus, des bits X arrivent sur la gauche

- Si a est non signé ou signé positif : $X = 0$
- Sinon (a négatif) : pas de norme

■ $\sim a$: *complément à 1*

- les bits de a sont inversés

Opérateur d'affectation

- *= est le symbole d'affectation :*
 - *lvalue = expression*
- *L'affectation est une expression => elle renvoie une valeur égale à la valeur de l'objet affecté*
- *Exemples :*
 - *i = 1; => affecte la valeur 1 à la variable i et renvoie 1*
 - *j = i + 1; => affecte la valeur 2 à la variable j et renvoie 2*
 - *while ((c = getchar()) != EOF)*

Opérateur d'affectation

- On peut combiner l'affectation avec l'un des dix opérateurs suivants : $+$ $-$ $*$ $/$ $\%$ $<<$ $>>$ $\&$ $|$ $^$
- Si op est l'un de ces dix opérateur alors :
 $expression1\ op=\ expression2$
est équivalent à
 $expression1 = expression1\ op\ expression2$
- Exemples :
 - $i += 3;$ est équivalent à $i = i + 3;$
 - $i /= j;$ est équivalent à $i = i / j;$

Opérateur conditionnel

- *`e1 ? e2:e3` est une expression qui vaut `e2` si `e1` est vrai et `e3` sinon*

- *Exemple :*

`taille<1.80 ? "petit" : "grand"`

*`printf("Il est %s", taille<1.80 ?
"petit" : "grand");`*

Opérateur de cast

- *(t) a*

- *Le cast sert à forcer le type d'un objet, c'est à dire convertir un objet d'un type vers un autre*

- *Exemples :*

 - (long int) x*y/(1.+sin(x))*

 - int carre(float x) {return (int) x*x;}*

Priorité et ordre d'évaluation

Opérateur	Associativité
() [] -> .	de gauche à droite
! ~ ++ -- - (t) * & sizeof	de droite à gauche
* / %	de gauche à droite
+ -	de gauche à droite
<< >>	de gauche à droite
< <= > >=	de gauche à droite
== !=	de gauche à droite
&	de gauche à droite
^	de gauche à droite
	de gauche à droite
&&	de gauche à droite
	de gauche à droite
?:	de droite à gauche
= += -= etc.	de droite à gauche

Priorité et ordre d'évaluation

- Certains choix de priorité sont plutôt mauvais (les concepteurs du langage en conviennent)
- La précedence des opérateurs bits à bits (c-a-d les opérateurs binaires : $\&$, \wedge et \mid) est plus petite que celle des opérateurs de comparaison (comme $==$ et $!=$)
 - $((x \& \text{MASK}) == 0)$ différent de : $(x \& \text{MASK} == 0)$
correspond à : $x \& (\text{MASK} == 0)$
- La précedence des opérateurs de décalage est plus petite que celle des opérateurs de $+$ et $-$
 - $a \ll 4 + b$ différent de : $(a \ll 4) + b$
correspond à : $a \ll (4 + b)$

Priorité et ordre d'évaluation

- *Le langage C ne précise pas dans quel ordre sont évalués les opérandes d'un opérateur*
 - $x = f() + g();$
- *L'ordre dans lequel sont évalués les arguments d'une fonction n'est pas précisé*
 - `printf("%d %d", i++, i);`
- *Expression avec effets de bords*
 - $a[i] = ++i;$
 - $a[i] = f();$

Priorité et ordre d'évaluation

- *On en déduit que l'écriture des instructions dont le résultat dépend de l'ordre de l'évaluation est un exemple de mauvaise programmation*

Quatrième partie



Les principales instructions

Les instructions et les blocs

- Une expression de la forme `x=0` ou `i++` ou `printf(...)` devient une instruction quand elle est suivie d'un `;`
- Le `;` fait partie de l'instruction, c'est un terminateur et non un séparateur comme en Pascal
- Une instruction composée ou bloc est une suite d'instructions encadrée par `{` et `}`

if



- **Syntaxe :**
 - *if (expression) instruction*
- *Si expression est vraie (<>0) instruction est exécutée*
- *Attention au piège de l'affectation*

if ... else



- **Syntaxe :**

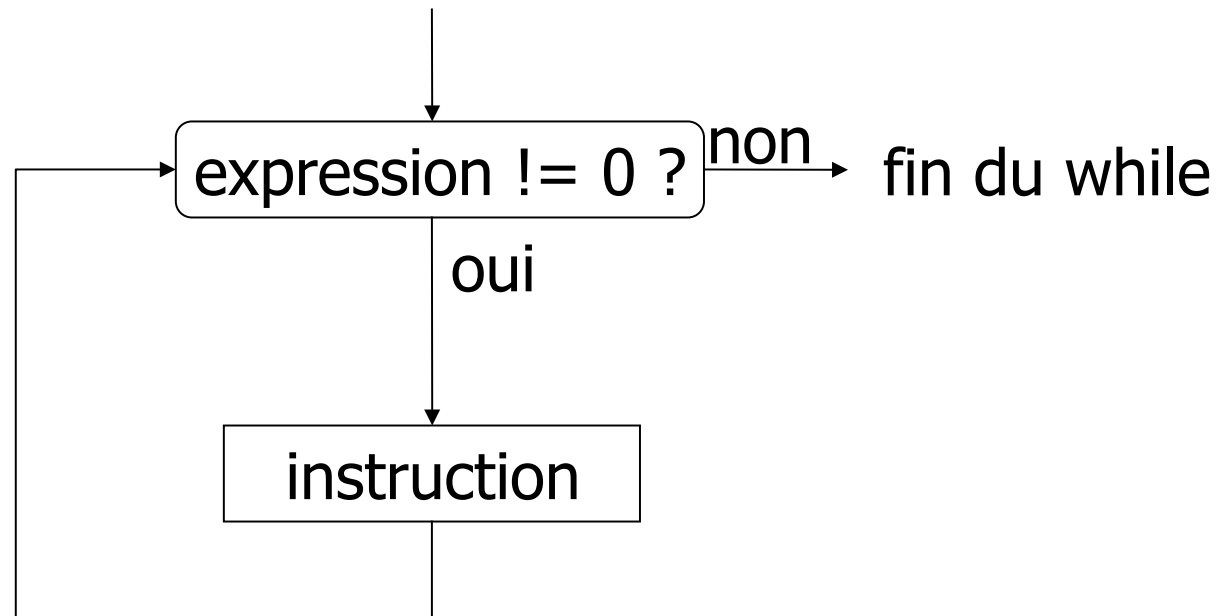
- *if (expression) instruction1 else instruction2*

- **Attention à la portée du else**

while

■ *Syntaxe :*

➤ *while (expression) instruction*



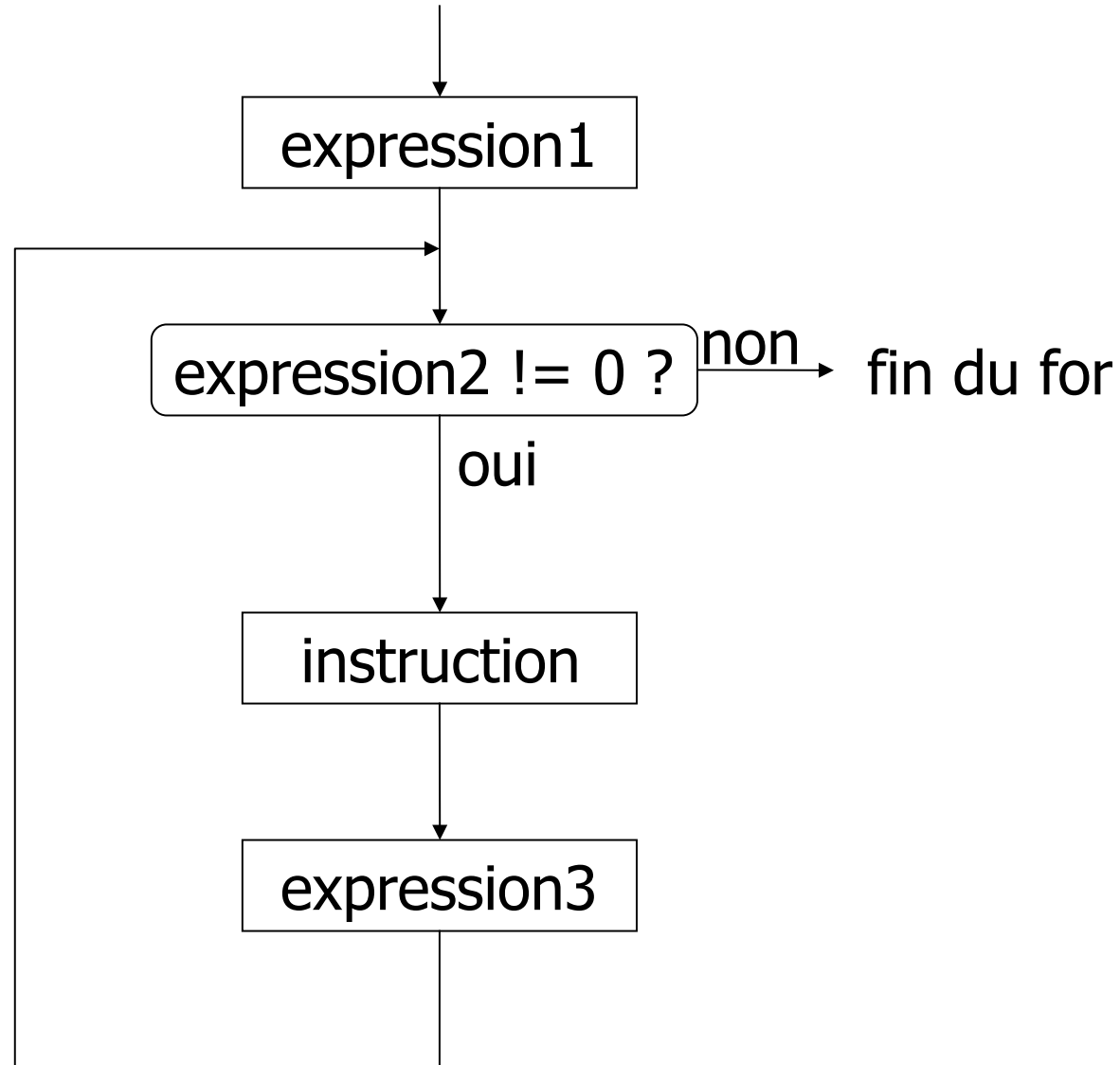
for



■ **Syntaxe :**

- *for (expression1; expression2; expression3)
instruction*
- *instruction est une instruction simple ou composée*
- *expression1 sert à initialiser*
- *expression2 est la condition de rebouclage*
- *expression3 est l'expression d'incrément*

for



for



- *Le for est un tantque traditionnel des autres langages.*
- *le for peut dans la plupart des cas être réécrit de la façon suivante :*
 - *expression1*
while (expression2)
{
instruction
expression3;
}

for



- *Les expressions peuvent comporter plusieurs instructions*
 - *exemple*
- *Rien n'oblige en C la présence des trois expressions :*
 - *for(;;) est valide et équivalent à while (1)*

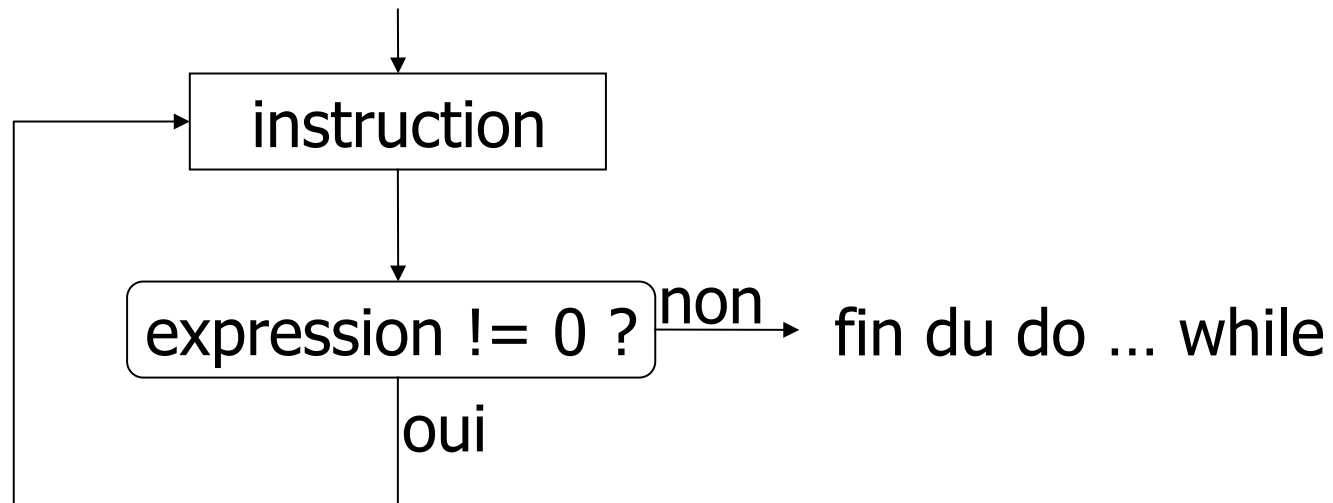
do ... while

■ *Syntaxe :*

➤ *do instruction while (expression);*

■ *do ... while est équivalent à :*

➤ *instruction*
while (expression) instruction



break



- *Cette instruction provoque la fin de l'instruction switch, while, do ou for qui la contient (au premier niveau)*
- *Elle est à utiliser avec précaution et devra toujours être justifiée par des commentaires*

continue



- *Cette instruction a pour but de provoquer le rebouclage immédiat de la boucle do, while, ou for qui la contient*
- *Dans le cas d'un for(e1;e2;e3), e3 est évaluée avant le rebouclage. C'est pour cette raison que l'équivalence entre le for et le while n'est pas totale.*

return



■ ***return;***

- *L'exécution de la fonction qui contient le return est interrompu, le contrôle est rendu à la fonction appelante.*

■ ***return expression;***

- *Idem avec une valeur de retour égale à l'évaluation de l'expression.*

goto étiquette;



■ *NE PAS UTILISER SOUS PEINE DE MORT !!!*

switch



■ *Syntaxe :*

```
➤ switch (expression)  
  {  
    case c1 : instructions  
    ...  
    default : instructions  
  }
```

■ *L'expression et les différentes constantes (ci) doivent être de type entiers, ou entiers définis par énumération.*

switch

- *Cette instruction est différente du case Pascal, car les valeurs de la constante sont vue comme des étiquettes de point d'entrée :*

```
➤ switch (i)
{
    case 1 : a=s[j];
    case 2 : b++;
    case 3 : a=s[j-1];
}
```

switch

■ *Utilisation de l'instruction break :*

```
➤ switch (i)
{
    case 1 : a=s[j];
            break;
    case 2 : b++;
            break;
    case 3 : a=s[j-1];
}

```

■ *Il n'y a pas de possibilité de donner d'énumération de valeurs ou d'intervalles*