REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de M'sila
Faculté des Mathématiques et de l'Informatique
Département d'informatique

جامعة المسيلة
كلية الرياضيات والإعلام الآلي
قسم الإعلام الآلي

Chapitre 3:
Instructions conditionnelles

Algorithmique et structure de données

Présenté par: Dr. Benazi Makhlouf
Année universitaire: 2022/2023

Contenu du chapitre 02:

1. Introduction
2. Structure conditionnelle simple
3. Structure conditionnelle composée
4. Structure conditionnelle de choix multiple
5. Le branchement

1. Introduction

Le programme est un ensemble **d'instructions**. La plupart de ces instructions sont exécutées dans **l'ordre** dans lequel elles apparaissent.

Les structures de contrôles sont des instructions qui **modifient** le déroulement **séquentiel** du programme comme: structures **conditionnelles**, boucles, appels de fonction et les instructions de **sauts inconditionnels**.

Les structures conditionnelles:

Nous rencontrons souvent des cas où il faut décider si une instruction doit être exécutée ou non, selon que la condition est vraie ou non.

Il existent trois types structures conditionnelles:

- simple (**si alors**)
- complexe (**si alors sinon**)
- la structure à choix multiples (**selon**).

2. La Structure conditionnelle simple "**si alors**"

Il se compose de deux parties :

•**Condition** : une expression de type booléen dont la valeur est soit vrai soit faux.

•**Bloc d'instructions** : exécuté si la condition est vraie, ou ignoré si la condition est fausse.

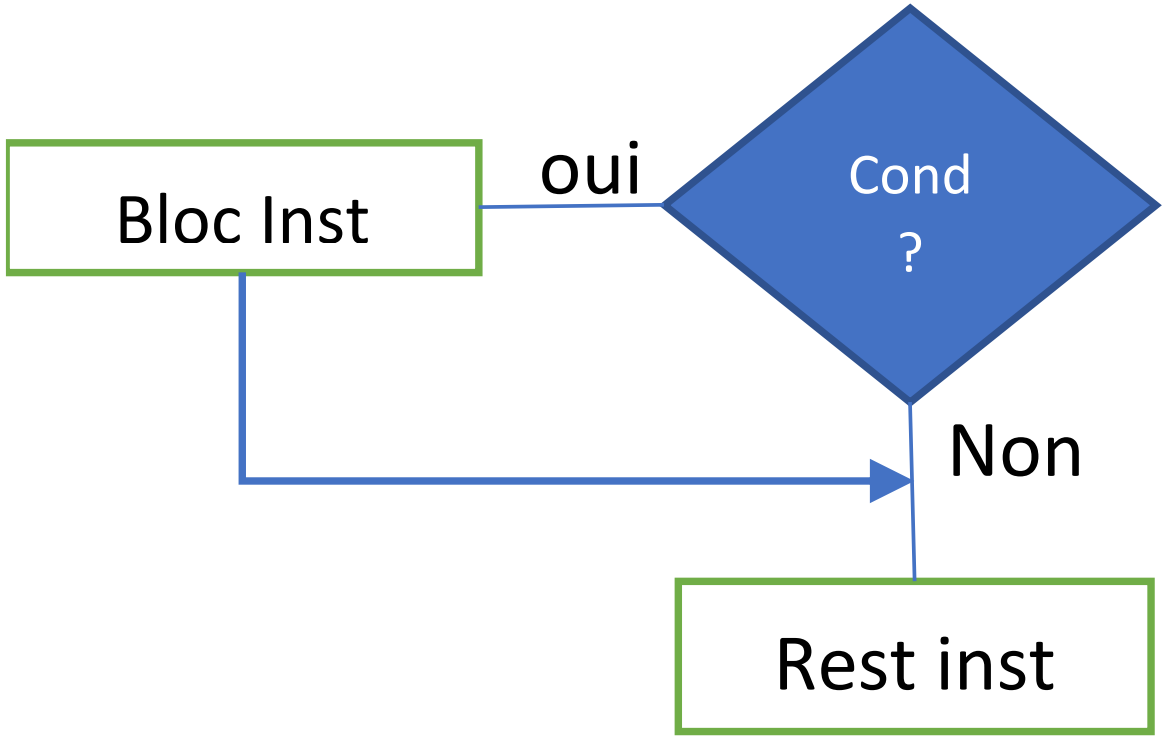
Syntaxe Algorithme

```
Si Condition Alors
  Un bloc d'instruction
FinSi
Le reste des instructions
```

Observations:

- **si** spécifie ce qu'il faut faire si la condition est vraie, mais pas ce qu'il faut faire si elle est fausse.
- **condition** est toujours entre les mots **si** et **alors**
- Pour construire la condition, nous utilisons des opérations de comparaison (>, <, = (==), ≠ (!=), ...) et des opérations logiques (et (&&), ou(||), non(!), ...).

Algorigramme



Langage C

Syntaxe

```
if (Condition)
{
   Un bloc d'instruction
}
Le reste des instructions
```

Observations:

- La **condition** est toujours entre parenthèses **()**.
- Les instructions qui appartiennent à **if** en **C** sont entourées de deux accolades **{}**
- accolades **{}** peuvent être **omises** si elles ne contiennent qu'une **seule** instruction.
- En **C**, le type **booléen** est exprimé par un **int**. Où **faux** est exprimé par **0** et **vrai** est n'importe quel nombre autre que 0. (**≠ 0**)
- Il n’y a pas de « **;** » après **}**.

Exemple

Écrivez un programme qui lit un entier, puis affiche un avertissement s'il est négatif, puis nous montre son carré.

Algorithme	C
algorithme racine **var** x :entier **Début** écrire("entrer un nbr") lire(x) **Si** x<0 **Alors** écrire("nbr est négatif") **FinSi** écrire("le carré est " , x*x) **Fin**	**#include** <stdio.h> int main() { int x ; printf("entrer un nbr \n") ; scanf("%d", &x) ; **if (x<0)** **{**// peut être dispensé printf("nbr est négatif \n") ; **}** printf("le carré est %d" , x*x) ; }

La Structure conditionnelle composée "si alors sinon"

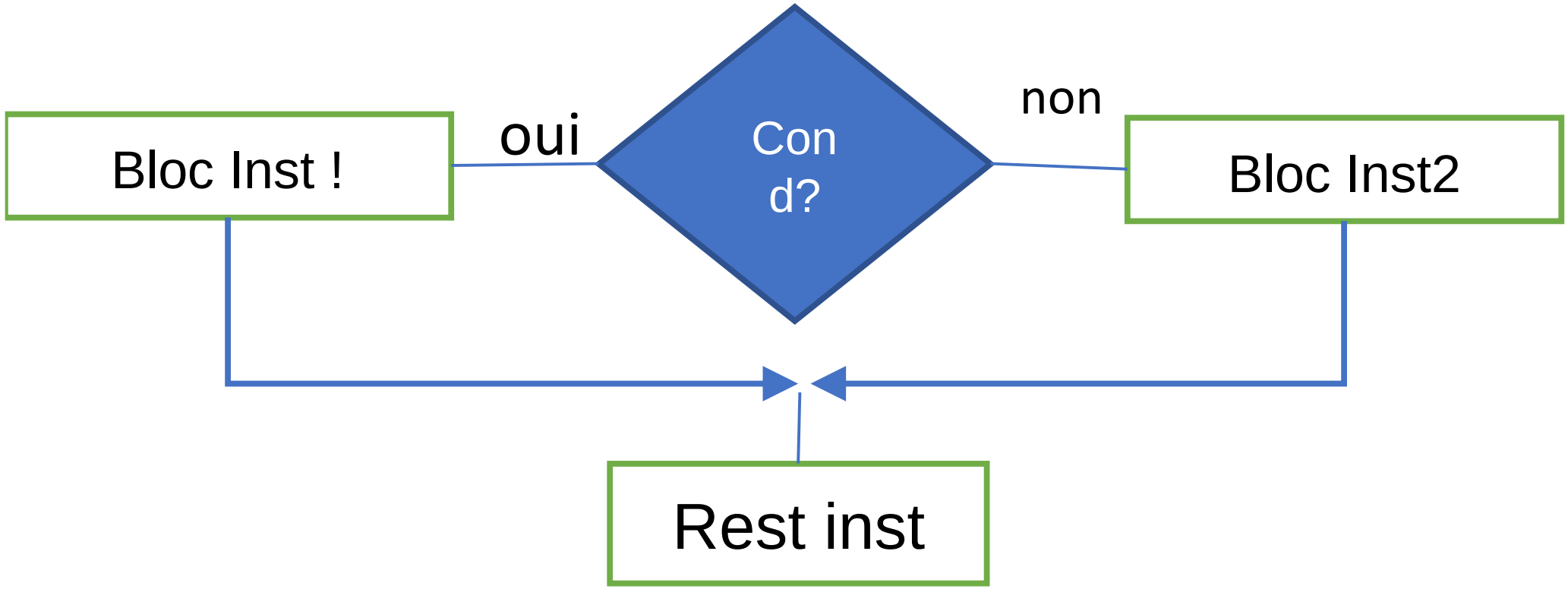
• la structure **si sinon** (**si.. alors.. sinon**...) qui est une extension de **si** simple. L'instruction conditionnelle composée **si sinon** comporte trois parties :

- Condition : une expression de type booléen dont la valeur est vraie ou fausse.
- Premier bloc d'instructions : Il est exécuté si la condition est vraie, ou ignorée si la condition est fausse.
- Deuxième bloc d'instructions : Il est exécuté si la condition est fausse, ou ignorée si la condition est vraie condition vraie.

Syntaxe

Algorithme	C
Si Condition **Alors** bloc d'instruction 1 **sinon** bloc d'instruction 2 **FinSi** Le reste des instructions	**if (**Condition**)** **{** bloc d'instruction 1 **}** **else** **{** bloc d'instruction 2 **}** Le reste des instructions

Algorigramme



Exemple

Écrivez un programme qui calcule la valeur absolue d'un nombre entier, puis l'affiche à l'écran.

Algorithme

```
algorithme absolue
var x, y :entier
début
 écrire("entrer un nbr")
 lire(x)
 Si x>=0 Alors
  y←x
 sinon
  y←-x
 FinSi
 écrire("|" , x ,"|=",y)
fin
```

C

```
#include <stdio.h>
int main(){
 int x, y ;
 printf("entrer un nbr \n") ;
 scanf("%d", &x) ;
 if ( x>=0 )
 { // peut être supprimé
   y=x ;
 }
 else
 { // peut être supprimé
   y=-x ;
 }
 printf(("|%d|=%d", x, y)) ;
}
```

```
if ( x>=0 )
  y=x ;
else
  y=-x ;
```

Affectation conditionnelle en C

Si nous avons une variable v prend l'une des valeurs v1 ou v2 selon la condition b, c'est-à-dire :

```
if (b)
  v=v1 ;
else
  v=v2 ;
```

Dans ce cas, l'opération ? : peut être utilisée et sa syntaxe est comme suit :

```
condition ? expression_vrai : expression_faut
```

- **condition** est une condition de type booléen
- **expression_vrai** L'expression renvoyée si la condition est vraie.
- **expression_faut** L'expression renvoyée si la condition est fausse

Exemple

- `v=b ? v1 : v2 ;`
- `result= moy>=10 ? "Admis" : "Ajourné" ;`

Extension de si sinon

Si sinon peut être utilisé pour tester plusieurs conditions et pour sélectionner le traitement approprié pour chaque cas. Par exemple : pour savoir si l'étudiant est admis ou non, il existe plusieurs cas. Soit il est admis sans compensation, soit il est admis avec compensation, soit il est admis avec des dettes, soit il est ajourné. Pour le savoir, il faut regarder les moyennes des premier et deuxième semestres s1 et s2, la moyenne annuelle MA et le total des crédits obtenus (Crd).

```
Si s1>=10 et s2>=10 Alors
 écrire("admis sans compensation ")
sinon
 Si MA>=10 Alors
  écrire("admis avec compensation ")
 sinon
  Si Crd>=45 Alors
   écrire("admis avec dettes ")
  sinon
   écrire("ajourné ")
  FinSi
 FinSi
FinSi
```

```
if ( s1>=10 && s2>=10 )
  printf("admis sans compensation ") ;
else if ( MA>=10 )
  printf("admis avec compensation ") ;
else if ( Crd>=45 )
  printf("admis avec dettes ") ;
else
  printf("ajourné ") ;
```

La Structure conditionnelle de choix multiple "selon"

Le test **selon** : est un cas particulier de l’instruction **si sinon** imbriqués. Il permet de déterminer le bloc à exécuter en fonction de la valeur d’une seule expression. On l’utilise si on teste une expression plusieurs fois.

selon, nous permet de rendre le programme plus lisible.

Il se compose de :

- L'expression à tester: est de type entier ou caractère Habituellement, c'est une variable. Par exemple : age.
- Les valeurs à tester avec le bloc d'instructions de chaque valeur.
- Un bloc d'instructions facultatif, qui s'exécute si la valeur actuelle ne correspond à aucune des valeurs mentionnées précédemment.

Syntaxe

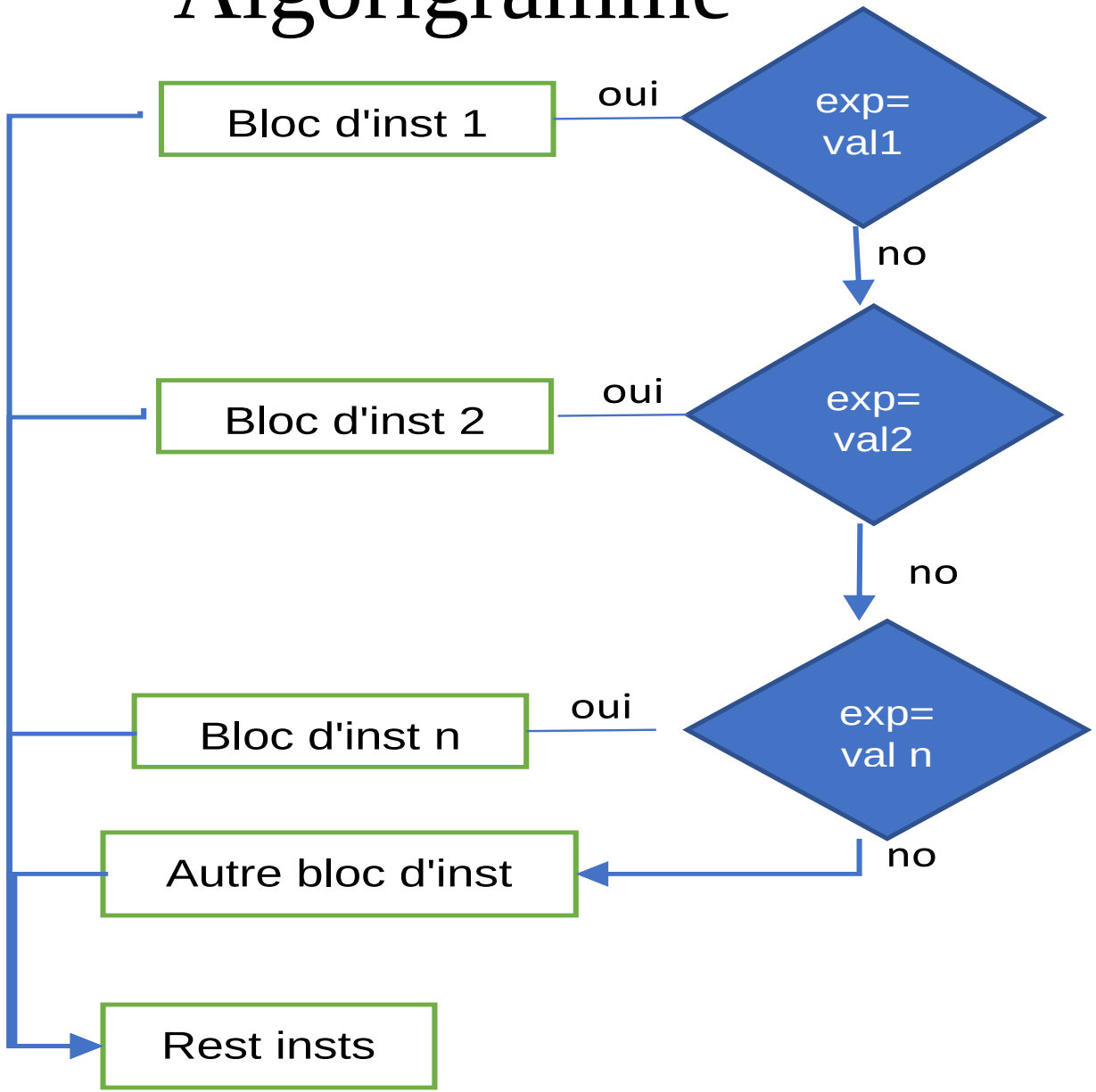
Algorithme	C
selon expression **faire** **case** val_1 **:** bloc d'instruction 1 **case** val_2 **:** bloc d'instruction 2 … **case** val_n **:** bloc d'instruction n **sinon** un autre bloc d'instruction **FinSelon** Le reste	**switch (** expression **)** { **case** val_1 **:** bloc d'instruction 1 **break ;** … **case** val_n **:** bloc d'instruction n; **break ;** **default:** un autre bloc d'instruction **}** Le reste

–**expression** : calculée pour obtenir une valeur de type entier ou caractère. Généralement, une variable.

–**val_1, ..., val_n** : une valeur ou une constante du même type que l'expression.

–**Bloc d'instruction** : une ou plusieurs instructions qui sont exécutées si la valeur de l'expression correspond à val_i.

Algorigramme



Les règles consternants switch

L'exécution **switch** en C diffère légèrement de selon, en ce que, après que le bloc de val_i a été exécuté, et que l'exécution ne rencontre pas l'instruction **break** ;, elle continuera à exécuter le bloc qui la suit, jusqu'à ce qu'elle rencontre la **break** ; L'exécution passe au reste des instructions en dehors du switch .
Pour que **switch** soit équivalent à selon, il faut ajouter **break** ; à la fin de chaque bloc.

- Les accolades {} du **switch** et les parenthèses () sont nécessaires et ne peuvent pas être supprimés.
- Chaque valeur val_i doit être différente de l'autre. Par exemple, il est illégal d'écrire le cas 1 deux fois.
- Le cas val_i peut être placé dans n'importe quel ordre. Cependant, il est recommandé de les placer par ordre croissant. Cela augmente la lisibilité du programme.
- Un bloc d'instructions peut être n'importe quel nombre d'instructions et de n'importe quel type.
- L'instruction break ; optionnel. Il est utilisé pour terminer un **switch** directement, déplaçant le flux de programme hors du **switch**.
- Le bloc **default** est facultatif. Si aucun cas val_i ne correspond, le contexte d'exécution est déplacé vers le bloc **default**. Ce doit être le dernier cas.

Si deux valeurs ou plus ont le même bloc d'instructions, l'algorithme peut utiliser une virgule. Alors qu'en C nous laissons la première valeur sans instructions ni **break** ;.

Exemple

L'algorithme	C
`case 7 ,9 : bloc d'instruction`	`case 7 :` `case 9 :` `  bloc d'instruction` `  break ;`

Exemple

Écrivez le programme qui lit un entier inférieur à 10, puis ce nombre apparaît en lettre à l'écran en anglais.

Algorithme

```
algorithme conversion
var nb :entier
début
 écrire("entrez un nbr")
 lire(nb)
 Selon nb faire
 case 0 : écrire("zero")
 case 1 : écrire("one")
 case 2 : écrire("two")
 …
 case 9 : écrire("nine")
 sinon
   écrire("not treated")
 FinSelon
fin
```

C

```
#include <stdio.h>
int main(){
  int nb ;
  printf("entrez un nbr \n") ;
  scanf("%d", &nb ) ;
  switch ( nb ) {
  case 0 : printf("zero") ;
           break ;
  case 1 : printf("one") ;
           break ;
  …
  case 9 : printf("nine") ;
          break ;
  default:
          printf("not treated") ;
  }
  return 0 ;
}
```

Exemple

Écrivez le programme qui lit un entier inférieur à 10, puis ce nombre apparaît en lettre à l'écran en anglais.

Algorithme

```
algorithme conversion
var nb :entier
début
 écrire("entrez un nbr")
 lire(nb)
 si nb=0 alors
 écrire("zero")
 Sinon
 si nb=1 alors
  écrire("one")
 Sinon
 si nb=2 alors
 écrire("two")
 …
 sinon
   écrire("not treated")
 FinSelon
FinSelon
```

C

```
#include <stdio.h>
int main(){
  int nb ;
  printf("entrez un nbr \n") ;
  scanf("%d", &nb ) ;
  if ( nb==0 )
  printf("zero") ;
  else if ( nb==1 )
  printf("one") ;
    …
  else if ( nb==9 )
  printf("nine") ;
  else
  printf("not treated") ;
  return 0 ;
}
```

Les Instructions de Branchement

C’est le processus de déplacement entre les instructions de programme exécutées par le processeur, où il effectue de « **saut** » à une adresse spécifique **au lieu** de continuer à exécuter des instructions **séquentiellement**.

Il y a quatre instructions en **C** qui peuvent modifier inconditionnellement le flux d'exécution d’un programme: **break**, **goto**, **continue** et **return**.

Instruction break;

- Permet de termines l'instruction "**switch**", déplaçant le flux vers la première instruction après "**switch**".
- Dans le cas d'une instruction "**switch**" imbriquée, elle ne sort que du "switch" auquel elle est directement liée.
- Il est également utilisé pour sortir des boucles. Dans ce cas, "**break** ; " est généralement à l'intérieur de "**if**".

Exemple

```
switch (grade){
  case 'A' :
  case 'a' : printf("excellent\n") ;
             break ;
  case 'b' : printf("good\n") ;
  case 'c' : printf("you can do better\n") ;
             break ;
  default : printf("try again\n") ;
}
```

Instruction **goto**

L'exécution du programme s'est transformée vers une instruction nommée.

Toute instruction peut être nommée avec un ID valide ("étiquette" ou "label") et deux points ":" devant elle.

```
label : instruction;
```

Pour accéder à cette instruction de n'importe où, nous utilisons la syntaxe suivante :

```
goto label ;
```

Exemple

```
ici : printf("zero") ;
```

Pour accéder à l'instruction ici de n'importe où, nous utilisons :

```
goto ici ;
```

Instruction **continue;**

Il est utilisé avec des boucles et permet au flux d'être déplacé vers la fin de la boucle et de passer directement à l'itération suivante, sans terminer la boucle.

Instruction **return**

Il est utilisé pour quitter les fonctions (semestre 2) et renvoyer le résultat.

Syntaxe

```
return expression ;
```

Exemple

```
return 0 ;
```

Fin Chapitre 03