Suite chapitre 2 Introduction au langage C (partie 5 Les opérateurs)

5.4. Les opérateurs d'affectation

Opérateur d'affectation =

variable = *expression* ;

l'expression de droite est évaluée puis stockée dans la variable

```
x = 3 ;
m = (c >= 'a') && (c <= 'z');
x = y = z = 0; de la droite vers la gauche.
```

variable op= expression

équivalent à *variable = variable op (expression)*

+=
-=
*=
\=
%=

ex : j *= i+1 ⇒ j = j * (i + 1)

Opérateurs d'incrémentation et de décrémentation :

++ incrémente de 1 la variable
-- décrémente de 1 la variable

++x et x++ ⇒ x = x+ 1 ;
--x et x-- ⇒ x = x - 1 ;

En préfixe : --x ou ++x la variable est incrémentée ou décrémentée avant son utilisation.
En suffixe : x++ x -- la variable est incrémentée ou décrémentée après son utilisation.

5.5. Les opérateurs d'adressage (vus dans le chapitre suivant)

5.6. Ordre de priorité des opérateurs (vu en TD)

forte priorité

Opérateurs	*Evaluation*
`() [] -> .`	gauche -droite
`! ~ ++ -- (<type>) * & sizeof`	droite - gauche
`* / %`	gauche -droite
`+ -`	gauche -droite
`<< >>`	gauche -droite
`< <= > >=`	gauche -droite
`== !=`	gauche -droite
`&`	gauche -droite
`^`	gauche -droite
`|`	gauche -droite
`&&`	gauche -droite
`||`	gauche -droite
`? :`	droite - gauche
`= += -= *= /= %= <<= >>= &= ^= |=`	droite - gauche

faible priorité

pour ceux de même niveau ⇒ de gauche à droite.

6. *Les instructions*

Instructions :

* Affectation
* Groupe d'instruction (bloc)
* Alternative
* Itération
* ~~Activation d'une tâche~~

6.1. Affectation

une affectation est une expression en langage C

C'est la seule action qui modifie l'état des variables.

variable = expression ;

instruction

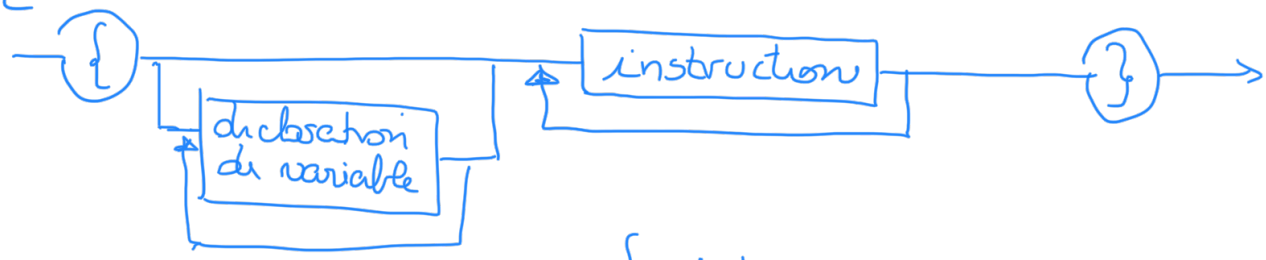
A-expression : expression qui contient une affectation

l'expression à droite du = est généralement du même type que la variable sinon il peut y avoir conversion implicite de type.

6.2. Séquence d'instructions (bloc)

instruction

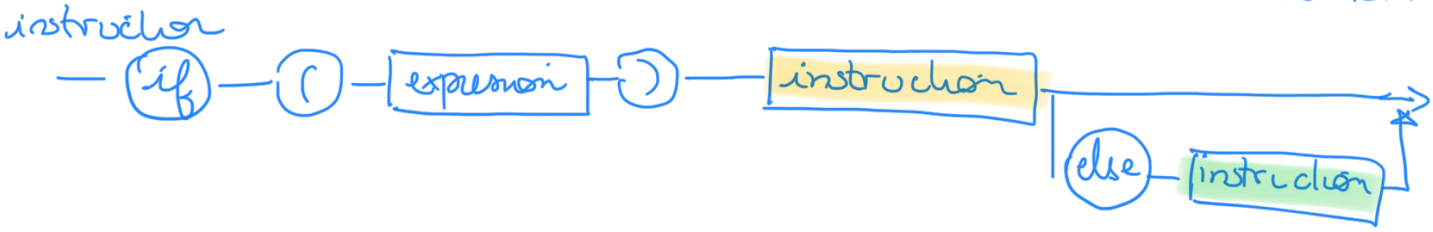
bloc




```
{ int x;
  int y = 0;
  x = y + 1;
  y--;
}
```

6.3. Alternative : if ... else

traduction du si alors sinon



Objectif ⇒ Rendre l'exécution d'une action dépendante d'une condition.

Principe : évaluer l'expression entre ( ) ;
si elle est vraie (= 1) alors l'instruction qui suit est exécutée si elle est fausse (⇒) alors l'instruction qui suit le else est exécutée.

L'alternative simple (sans else) est possible.

En cas d'alternatives imbriquées, un else se rapporte toujours au dernier if rencontré si celui-ci n'est pas terminé.

Exemples :

```
int x, y, min ;
if (x < y)
    min = x ;
else
    min = y ;
```

→ affectation

→ affectation

int x,y; /* données */
int min; /* resultat */

int x;
int vabs;

vabs = x;
if (x < 0)
vabs = -x;

```
#include <stdio.h>

int main ()
{
    char caractere;

    printf("etes vous pour ou contre ? [o/n]:");
    scanf("%c", &caractere);

    if ((caractere == 'o') || (caractere == 'O'))
        printf("\nc'est OUI\n");
    else
        if ((caractere == 'n') || (caractere == 'N'))
            printf("\nc'est NON\n");
    else
        printf("\nc'est JE NE SAIS PAS\n");

    return 0;

}
```

lire (caractère)

Calcul de x^n $n \geq 0$

```
#include <stdio.h>
int main()
{ float x; /* donnée */
  int n; /* donnée >=0 */
  float P; /* resultat */
  int i; /* compteur */
  printf("donner la valeur x et n (>=0) : ");
  scanf("%f %d", &x, &n);

  P = 1;
  i = 0;

  while (i != n) {
      P = P * x;
      i = i + 1;
  }
  printf("%f puissance %d = %f", x, n, P);
  return 0;
}
```

initialisation du compteur

test

incrément du compteur

6.4. Itération (Boucles)

3 types de boucles

POUR	⇒ FOR
TANT QUE	⇒ WHILE
REPETER	⇒ DO WHILE

1- La boucle : while ()

Diagramme de conway

Objectif : répéter une instruction tant que l'expression est vraie.

Principe : 1. - Evaluer l'expression entre ( )
2. - Si celle-ci est vraie (non nulle) alors l'instruction est exécutée. On évalue de nouveau l'expression et on recommence tant que l'expression est entre () vraie.

Exemples

Programme qui calcule le reste de la division entière.

pour utiliser printf()
scanf()

```c
#include <stdio.h>

int main (  )

{
int x, y; /* données */
    int reste ; /* resultat : reste de la division entiere sans utiliser mod */

    x = 10 ;
    y =  3 ;
    reste = x;
    while (x > y)
        x = x - y ;
    printf (" le reste = % d "  , x) ;
return 0;

}
```

reste

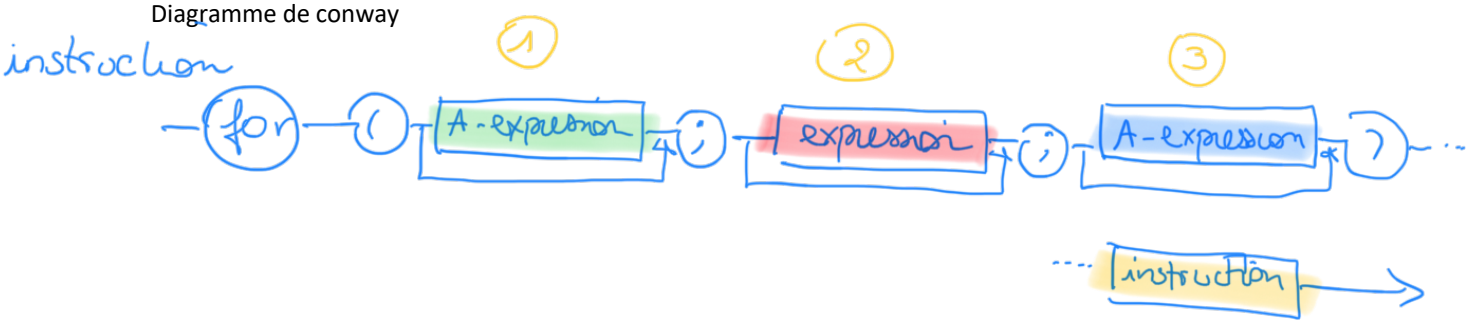
x = 10 ;
y = 3 ;
reste = x ;
while (reste > y)
reste -= y ;
printf (" le reste = %d ", reste);

Calcul du PGCD de deux entiers positifs

%f : float
%d : int
%c : char

La boucle for

Diagramme de conway



Expression1 et Expression3 sont des A - expressions c'est-à-dire des affectations

Expression2 est une expression dont on teste la valeur vraie ou fausse

Principe : 1 - on exécute l'expression1 ← 1 seule fois
2 - on évalue l'expression2
si elle est vraie
- on exécute l'instruction
- puis on exécute l'expression 3 et on recommence en 2-
sinon poursuivre après instruction.

ON utilise la boucle for lorsqu'on connait le nombre d'itérations (utilisation d'un compteur)

Exemples de boucle for

affichage de la table de 9

```
int n;
for (n = 1 ; n <= 10 ; n++)
    printf("%d * 9 = %d \n ", n, n * 9);
```

n : 1 2 3 .... 9 10 11

1 * 9 = 9
2 * 9 = 18
...
9 * 9 = 81
10 * 9 = 90

Calcul du reste de la division entière.

Calcul de x^n $n \geq 0$.

```
P = 1;
for (i = 0 ; i != n ; i++)
    P = P * x;
```

ou

```
for (P = 1, i = 0 ; i != n ; P *= x, i++);
```

autres écritures

```
for (P = 1, i = 0 ; i != n ; i++)
    P *= x;
```

ou

```
P = 1;
i = 0;
for ( ; i != n ; i++)
    P *= x;
```

4

la 1 est plus lisible

Plus général : Les 3 champs de l'instruction peuvent n'avoir aucun lien entre eux (très peu lisible)

```
for (i = j = 0, a = b ; (c = getchar ( )) ! = '\t' && c != '\n'; i++ , -- a)…
```

```
/* exemple boucle for */
    int i, n, produit;

    produit =1;
    for (i = 1 ; i<= MAX ; i ++)  {
        printf ("  Entrez un nombre  : ") ;
        scanf ("%d" , &n) ;
        produit *=  n ;
    }
    printf("\n le produit de ces nombres vaut %d ", produit);
```

3. - La boucle do while

Diagramme de conway

Principe :

1. Exécute l'instruction
2. Evalue l'expression entre ( )
   si elle est vraie on recommence l'exécution en 1.
   sinon poursuivre après le (;).

Remarque : l'instruction est exécutée au moins 1 fois.

Exemple

Programme qui calcule le reste de la division entière.

```
printf("donner 2 entier positifs SVP");
scanf("%d %d ", &x,&y);
reste = x;
do
    x = x - y ;
while (x > y);
printf (" le reste = % d "  , x) ;
```

quelle différence avec la boucle while ?

la boucle do while convient bien au test des hypothèses sur les variables entrées par l'utilisateur qui sont des données de l'algorithme

Exemple

```
do {
    printf("entrer une valeur positive ou nulle svp : ");
    scanf("%f", &x);
}
while (x<0);
```

Programme avec une boucle while exemple

Calcul de x^n $n \geq 0$

```
#include <stdio.h>
int main()
{ float x; /*donnée */
  int n; /* donnée >=0 */
  float P; /* resultat */
  int i; /* compteur */
  printf (" donner les valeurs x et n (>=0) : ");
  scanf ("%f %d", &x, &n);
```

initialisation du compteur

```
  P = 1;
  i = 0;
```

test

```
  while (i != n) {
      P = P*x;
      i = i+1;
  }
```

incrément du compteur

```
  printf ("%f puissance %d = %f",
                  x, n, P);
  return 0;
}
```