De même b-- post decrementation, --b pré-décrementation.

I) Expressions.

Il y a 3 catégories d'expressions
- arithmétique. (calcul)
- relationnelle (comparaison)
- logique (condition)

- Operateurs binaires

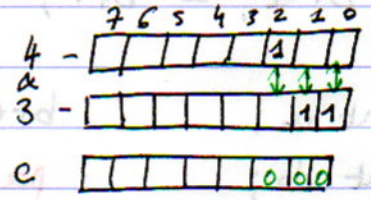
→ le principe

- Faire des calculs sur les bits contenus dans un octet, plus generalement un "mot" (groupe de bits indivisibles)
- Le calcul correspond à celui de l'algèbre de boole mais s'applique bit à bit sur tous les bits d'un mot.

Les opérateurs classique & - et (binaire), | ou (binaire), ^ ou exclusif.

exemple : calcul Bit à Bit.
(en C un char correspond à 1 octet)

```
{char c ;(un octet)
    c = 4&3 ; (c vaut 0)
    d = 2&3 ;
}
```



representat° binaire sur 1 octect de 4 et de 3.

$n = \sum_{i=0}^{j} a_i 2^i$.

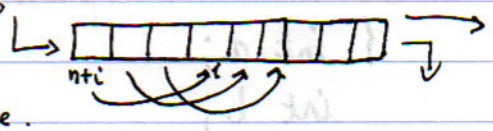
4&3 = 0, 2&3 = 2.
4|3 = 7, 2|3 = 3.
4^3 = 7, 2^3 = 1.
negation ~ : convertit 1 en 0 et 0 en 1.

- Operateur calculant sur le profil binaire.

décalage -

>> decalage logique droit.

x >> n.



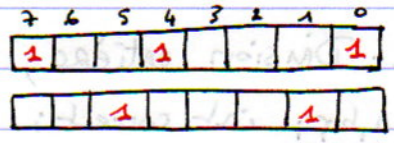
- on décale à droite n fixe.
- sur le profil binaire à gauche, on ajoute des 0.
- les bits décalés qui n'ont pas leur place seront supprimés.

x << n décalage logique gauche.

- décalage à gauche de n fixe.
- ajout de 0 à droite
- perte des bits le + à gauche

ex : 4 >> 1 = 2.

```
{char c;
c = 145;   (c = 33)
c = c << 1;  (c = 2)
c = c >> 4;
}
```



$2^7 + 2^4 + 2^1$

$128 + 16 + 1 = 145$

$2^5 + 2^1 = 34$

Application :

Dans un ordinateur, la division coûte n 10 fois plus de temps que l'addition ou des opérateurs binaires.
En algorithmique, la division par 2 sur les entiers est fréquente.
On utilise le décalage logique pour faire la division par 2 dans $\mathbb{N}$.

$x / 2^i \equiv x >> i$, $x \in \mathbb{N}$.

Expression Relationnelle.

En C les opérateurs retournent les valeurs 0 ou 1.
Donc on pourrait l'utiliser pour un calcul arithmétique.
ex (mauvais) : $(2<3) + (2<5)$ conseil : A NE PAS FAIRE !
$1 + 1 = 2$

Opérateur logique

- les expressions logiques retournent les valeurs 0 ou 1.
- l'évaluation des opérateurs logiques est "paresseuse".
  on se contente du minimum de calcul pour avoir le résultat.

exemple : {int a,b,c;
a = 0;
b = ...;
c = a && b ; c vaut 0 après b.

Lors de l'évaluation paresseuse, on évalue d'abord le premier argument (à gauche) et si la valeur permet d'avoir le résultat, le calcul du 2nd argument N'EST PAS FAIT.

ex : 1 || x → 1.

0 && x → 0.

L'ordre des arguments a de l'importance.

exemple : Division entière, (1/0 erreur) avec vérification d'erreur.

```
{ int a,b,c; int correct;
  a= ...;
  b= ;
  correct = (b!=0) && ((c = a/b) || 1);
```

La division c=a/b ne se fera que lorsque b!=0.

a=5; b=2

correct = (b!=0) && (1 || (c=a/b)) ⚠ ATTENTION A L'ORDRE

1 → Ce calcul ne s'effectue jamais car 1 || x = 1.

Constante définie symboliquement.

- permet de définir des valeurs par des symboles.

Symbodes

ex: π, g, taille quelconque.

- pour définir ces constantes, on utilise #define

ex : #define PI 3.1415 (majuscule)

```
main ()
{ float v;
  float r;
  v = (4.0/3.0) * PI * r * r;
}
```

une constante symbolique ne peut se modifier.

lors de la compilation il existe une phase de preprocessing qui remplace les constantes (partie gauche) par les valeurs (partie droite).

Autre utilisation du #define définir un nouveau langage.

```
#define PRINCIPAL main        #define FIN }
#define REEL float            PRINCIPAL ()
#define DEBUT {               DEBUT
```

Priorité - Associativité.

Permet de donner un ordre non ambigüe sur l'évaluat° des expressions.

3+4/7 ; 1 ou 3 selon l'ordre du calcul.

(3+4)/7 ; → 1 et 3+(4/7) ; → 0

- pour lever ces ambigüité, on donne des priorités de calcul sur les operateurs.
- ceci correspond à une relation d'ordre sur les operateurs (à connaître)

/ est prioritaire sur +. le resultat est 3.

++ (--) sont de + hautes priorités.

associativité.

- définie un ordre sur les operateurs de même priorité
- cela se définit par rapport à un même besoin de calcul.

30/2/2 2 résultats possibles 30 ou 7 30/(2/2) → 30 ou (30/2)/2 → 7.

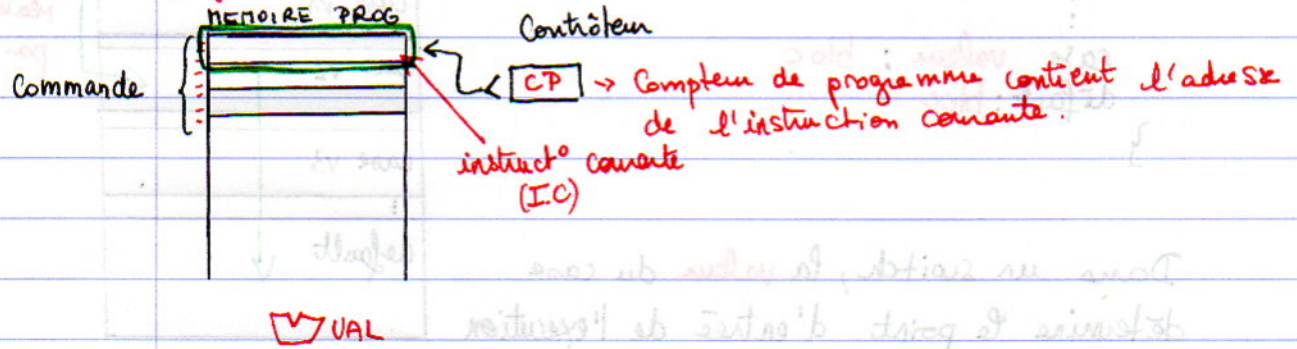
l'associativité à gauche signifie que le calcul doit être fait pour le 1er argument lorsque l'expression est évaluée.

(/ est associatif à gauche)

l'associativité à droite implique le calcul du 2ème argument (celui de droite)

VI) Instruction.

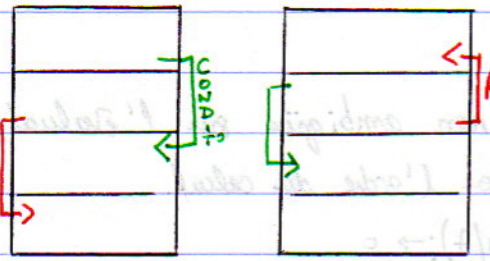
• les instructions permettent de contrôler le flot d'execution du programme.



Il y a 2 types de contrôle : - contrôle du pt de vue de l'execution
ie modif. de CP.

2 Types de STRUCTURES DE CONTRÔLE :

- la condition.
- l'itération ou la répétition.



CONDITION (if) — ITERATION (while)

Branchement inconditionnel

Branchement Conditionel qui dépend du résultat d'une Expression logique.

Structures Conditionnelles.

But : définir une exécution d'un bloc conditionné au résultat d'une expression logique.

2 structures conditionnelles :

if - switch

syntaxe

```
if (condition)
{ Bloc si vrai }
else
{ Bloc si faux }
```

ex : la valeur absolue |x|

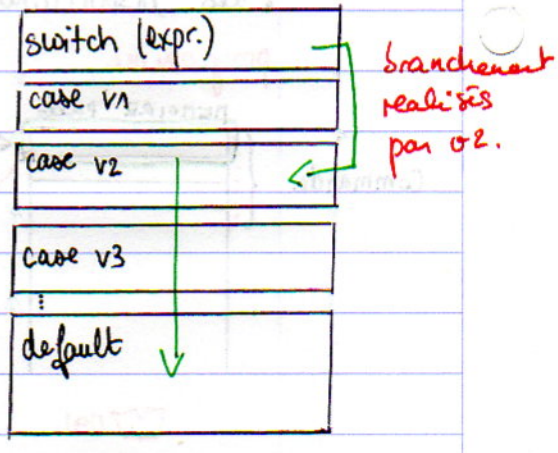
```
{int x, abs;
 x= ... ;
 if (x>0)
   {abs =x;}
 else
  {abs = -x;}
}
```

```
{int abs, x ;
 x= ... ;
 abs = x ;
 if (abs < 0)
 { abs = -abs;}
}
```

Le switch représente un choix multiple :

```
switch (expr.)
{
  case valeur : bloc
   :
  case valeur : bloc
  défaut : bloc
}
```

PROG. EN MEMOIRE



Dans un switch, la valeur du case détermine le point d'entrée de l'execution des blocs associés à chaque case.

1 fois ce point d'entrée trouvé, l'execution poursuit en sequence et les blocs en dessous du pt d'entrée sont executés.

Exemple :

1	2	3	4	5	6	7	8	9	10	11	12
31	28/29	31	30	31	30	31	31	30	31	30	31

↑ 7 mai : 5 jours / mois avant mai + jours.

```
switch (m-1)        (m-1) => 4
{ case 12 : n = 31;
  case 11 : n+ = 30;
  case 10 : n+ = 31;
  case 9  : n+ = 30;
  case 8  : n+ = 31;
  case 7  : n+ = 31;
  case 6  : n+=30;
  case 5  : n+ = 31;
  case 4  : n+ = 30;   37
  case 3  : n+ = 31;   68
  case 2  : if (a%4 == 0)
              {n+ = 29;}   } partie fausse
            else {n+=28;}  96
  case 1  : n+=31;   127
printf ("le nombre de jours est %d", n);
```

```
main ()
{int n /* ensemble de jours */
 int j, m, a ;
 scanf ("%ad %d %d", &j, &m, &a);
 (lire j, m, a)
 n=j ;
```

on prend j=7, m=5, a=2005

Utilisation du case : Sous forme de Menus.

```
ex: main ()
    {int c;
ecrit -> printf ("voulez-vous 1) lire 2) ecrire 3) fermer");
lit -> scanf ("%d, &c);
       switch (c)
       { case 1 : {- LIRE; break ;}
         case 2 : {- ECRIRE; break ;}
         case 3 : {- FERMER; break ;}
         default : {printf ("erreur");}}
```

le break introduit un pt de rupture dans l'execution de la condition. Il a pour effet de faire parcourir l'execution après la structure conditionnelle dans lequel il est inséré.

Structuras de contrôle itératives.

But : définir la répetition d'un traitement. ou itérer un traitement.

en C : - 3 boucles différentes sont disponibles -

for, while, do... while.

• while : syntaxe. ← condition de poursuite du traitement.

```
while (<condition>)
  {<bloc>}
{
}
```
↖ traitement effectué à chaque étape.

sémantique (Signification) :

tant que la <condition> est verifiée (vrai), il est necessaire d'executer le traitement contenu dans le <bloc>.

* exemple. Calculer la somme des n premier entier dont le résultat est inférieur à une valeur $n \in \mathbb{N}$.

$$S = \sum_{i=0}^{k} i, \quad S \leq n.$$

① s=0; ② i=0; } → pré-traitement.

Tant que $S \leq n$ faire
  ajouter i à s (s=s+i)
  augmenter i de 1.
fin.

③ while (s <= n)
④ { i = i+1;
  { s = s+i;
  }
⑤ s = s-i; } → post-traitement.

Que vaut la variable s à la sortie? (s = n+i)

Déroulement de la boucle : execution tb décrivant les états des variables - le corps de boucle n'est pas executé.

n=0

	i	s	c
①	0	?	
②	0	0	
⑥			V
③	1	1	
⑥.2			F