n = 4.

		i	s	c	
	①	0	?		
	②	0	0		
>	ⓒ			V	
	③	1	0		itération i=1
	④	1	1		
>	ⓒ			V	
	③	2	1		iterat° i=2
	④	2	3		
>	ⓒ			V	
	③	3	3		iterat° i=3
	④	3	6		
	ⓒ			F	
	⑤	3	3		

- le tb décrivant l'execution s'nomme trace. Une trace est une représentation décrivant l'exec du prog.
- Une itération correspond à une exécution particulière de la boucle. Elle peut être assimilée à un morceau de Trace.
- Un compteur d'itération determine le nb d'itération.

Dans une structure itérative, on distingue :
- le pré-traitement séquences située avant la boucle.
- le corps " " à l'intérieur ".
- le post-traitement " " à l'extérieur ".

Version compacte

```
main ()
{ int s,i,n;
  scanf ("%d", &n);
  s = i = 0;
  while ((s += ++i) <= n);
  s -= i;
}
```

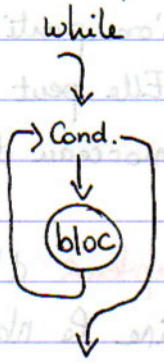
• Do... while

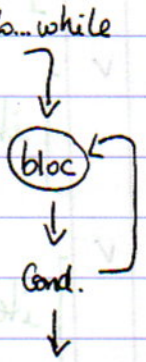
syntaxe

```
do
{bloc}
while (condition)
```

sémantique :
faire le traitement tant que la condition est vraie.

à la différence du while, le do... while commence à faire le traitement puis évalue la condition.






```
s=i=0;
do
i=i+1;
s=s+i;
while (s<=N)
s=s-i;
```

→ équivalence avec le while

```
<init>
do
<bloc>
while (<cond>)
<fin>
```

```
<init>
<bloc>
while <cond>
  <bloc>
<post>
```

```
s=i=1;
while (s<=n)
{ i=i+1;
  s=s+i;
}
s=s-i;
```

• FOR

syntaxe

```
for (<e1> ; <condition> ; <e2>)
   {<bloc>}
```

équivalence en while :

```
<e1>
while <condition>
{ <bloc> ;
} <e2>
```

→ exemple :

```
{ int i,s;
  i=1;
  for (s=   ; s<=n; s+=i) {i++;}
  s=s-i;
}
```

```
{int i,s,n;
i=s=0;
for(i=1;s<=n;i++)
  {s+=i;}
s=s-i-1;
```

```
s=0;
i=1;
while (s<=n)
{ s+=i;
  i++;
}
```

Utilisation classique du for -

Les champs du for (e1, cond, e2)
Servent usuellement à donner une valeur initiale à une variable,
établir une condition sur la variable,
définir un traitement de modification de la variable.
Ils (les champs) gèrent des compteurs.

→ exemple : Produit factoriel $n! = \prod_{i=1}^{n} i$.

```
main ()
{ int i, n;
  long int fact;
  } n ?
  fact = 1;
  for (i=1; i<=n; i++) { fact *= i; }
  printf("%d", fact);
}
```

Définir un algorithme à partir d'un problème.

Soit $n \in \mathbb{N}$,
$\exists ?\, n$ tq $n! = m$.
la réponse est oui ou non.

4 éléments à définir
* pré traitement ③
* traitement ① > calcul de factoriel.
* Condition ②
* post traitement ④

- Traitement correspondant au calcul de factoriel. Il se fonde sur une propté de récurrence.

on a une propriété qui est vraie à toutes les étapes.

$(n+1)! = n! \times (n+1)$

on a défini une transition entre 2 itérations successives de la boucle.

```
n=0;                    (fact = 0!)
fact=1;
while ( fact < m
{ n= n+1 ;
  fact = fact * n ;     (fact = n!)
}
```

k (n = k ; fact = k!

k+1 | n = k+1 ; fact = fact * n ; fact k! × (k+1) ; fact = k+1 !

condition d'arrêt, elle s'appuie sur la croissance de $n!$

fact : $\mathbb{N} \to \mathbb{N}$

fact est croissante ie

$\forall n \in \mathbb{N} \quad \forall m! \in \mathbb{N}$

$n \leq n! \Rightarrow n! \leq m!$

donc : $\forall m \in \mathbb{N} \quad \exists n \in \mathbb{N} \quad n! \leq m < n+1 !$

Dans le post traitement, il faut vérifier que le resultat correspond à la question posée.

```
{int n, fact, m;
 int test;
 n=0;
 fact = 1;
 Scanf("%d", &m);
 while ( fact < m
 { n=n+1;
   fact= fact * n; }
 test = ( fact == m);
}
```

STRUCTURES DE DONNÉES.

Le 03/10/2005.

• Ensemble énuméré

Il s'agit de la définition d'un type.
Il décrit un ensemble d'éléments.

→ exemple : ensemble de couleurs.

```
enum COULEUR {Rouge, Vert, Bleu}
```

toutes variables de ce type possède comme valeur un elt de cet ensemble.

```
main ()
{ enum COULEUR C;
   C = vert;
}
```

→ exemple 2 :

```
enum BOOL {false, true}
main ()
{ enum BOOL test;
  test = false;
}
```

typedef :
Permet de renommer la définition de type par un nom plus simple.

```
typedef enum COULEUR {rouge, vert, bleu} color_t;
typedef enum BOOL {false, true} bool_t;

main ()
{ bool_t test;
  test = false;
}
```

Comment sont représentés les elements de l'ensemble lors de la compilation ou de l'execution ?
Lors de l'execution les éléments (rouge, vert, bleu)

sont codés par des entiers avec la règle suivante

1er element → 0.

2ème " → 1.

i+1ème " → i.

Application -

- Comparaison des éléments par rapport à leur position rouge < vert.
- Utilisation dans les boucles

  ∀c ∈ Couleur,

  for (c = rouge, c <= bleu, c++)
- pour le type booléen, l'ordre importe.

typedef enum FOBOOL {true, false} bool_t
                    1   0

```
main ()
{ bool_t test;
  test = false ;  1     sortie 1 car false est en 2ème position
  if (test)             et vaut 1 (true 0).
  { printf("1") ; }
  else
  {printf("2") ; }
}
```

Enregistrement :

type - But : agréger dans un même type des données de type différents.

Déclaration de type :

struct NOM {(type : nom ;)*}

exemple : la définition des complexes.

c = a + bi

réel ↑ ↑ imaginaire

struct COMPLEX {float reel ; float img ;}