Algorithmique et Langage C

www.polytech.unice.fr/~vg/fr/enseignement/xidian

Granet Vincent - vg@unice.fr

Xi'an - Octobre 2015 - Mai 2024

POLYTECH NICE-SOPHIA

西安电子科技大学
XIDIAN UNIVERSITY

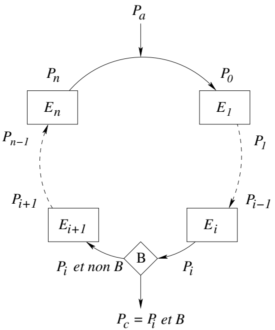
Sommaire

Énoncés Itératifs

Introduction

Les énoncés itératifs sont des énoncés qui permettent l'exécution répétitive d'un ou plusieurs autres énoncés (élémentaires ou structurés).

- On les appelle aussi boucles
- Les langages de programmation proposent plusieurs types d'énoncés itératifs : **tantque**, **répéter**, **pourtout**...



Règle de déduction

si $\{P_0\} \overset{E_1}{\Rightarrow} \{P_1\} \overset{E_2}{\Rightarrow} \{P_2\} ... \{P_{i-1}\} \overset{E_i}{\Rightarrow} \{P_i\} \overset{E_{i+1}}{\Rightarrow} \{P_{i+1}\} ... \{P_{n-1}\} \overset{E_n}{\Rightarrow} \{P_n\}$ et $\{P_n\} \Rightarrow \{P_0\}$ et $\{P_a\} \Rightarrow \{P_0\}$ et $\{P_i \text{ et } B\} \Rightarrow \{P_c\}$ et $\{P_i \text{ et non } B\} \overset{E_{i+1}}{\Rightarrow} \{P_{i+1}\}$
alors $\{P_a\}$ énoncé-itératif-général $\{P_c\}$

Invariant de boucle

- Toutes les affirmations $P_0 ... P_n$ sont *invariantes*
- L'affirmation P_i qui précède la condition d'arrêt B joue un rôle particulier. Elle est représentative de la sémantique de l'énoncé itératif, et elle est appelée l'**Invariant** de boucle.

Finitude

- Pour que la boucle s'achève, la condition d'arrêt doit être vérifiée.
- À démontrer formellement. On cherche une fonction $f(X_B)$ des variables modifiées dans la boucle et utilisées par B, stritement *positive* et *décroissante* vers une valeur particulière (*e.g.* 0) elle que la condition d'arrêt soit vérifiée.

L'énoncé tantque

- Lorsque les énoncés E_1 à E_i sont vides
- au minimum, zéro itération
- P est l'invariant

{P} **tantque** B **faire** E **fintantque** {P et non B}

Règle de déduction

si $\{P\ et\ B\} \overset{E}{\Rightarrow} \{P\}$
alors $\{P\}$ **tantque** B **faire** E **fintantque** $\{P\ et\ non\ B\}$

Exemple : factorielle, calcul croissant

- $0! = 1$ et $n! = 1 \times 2 \times 3 \times \cdots \times (i-1) \times i \times \cdots \times n$
- *Finitude* : $f(i) = n - i$

```
{Antécédent : n≥0}
{Conséquent : factorielle(n) = n!}
fonction factorielle(donnée n : naturel) : naturel
variables
   i, fact type naturel
   i ← 0
   fact ← 1 {Invariant : fact = i!}
   tantque i<n faire
      {fact × (i+1) = i! × (i+1) = (i+1)! et i<n}
      i ← i+1
      {fact × i = i!}
      fact ← fact×i
      {fact = i!}
   fintantque
   {i = n et fact = i! = n!}
   rendre fact
finfonc {factorielle}
```

Exemple : division entière

- $\forall a \geqslant 0,\ b > 0,\ a = \text{quotient} \times b + \text{reste}\ ,\ 0 \leqslant r < b$
- *Finitude* : $f(reste) = reste - b$

```
{Antécédent : a⩾0, b>0}
{Conséquent : a = quotient×b + reste, 0 ⩽reste<b}
procédure DivisionEntière(données a, b : naturel
                          résultats quotient, reste : naturel)
  reste ← a
  quotient ← 0
  {Invariant : a = quotient × b + reste}
  tantque reste ⩾ b faire
    {a = (quotient+1) × b + reste − b et reste⩾b}
    quotient ← quotient+1
    {a = quotient × b + reste − b et reste⩾b}
    reste ← reste−b
    {a = quotient × b + reste}
  fintantque
  {a = quotient × b + reste et 0⩽reste<b}
finproc {DivisionEntière}
```

L'énoncé répéter

- Lorsque les énoncés E_{i+1} à E_n sont vides
- au minimum, une itération
- Q est l'invariant

{P} **répéter** E **jusqu'à** B {Q et B}

Règle de déduction

si $\{P\} \overset{E}{\Rightarrow} \{Q\}$ et $\{Q \textit{ et non } B\} \overset{E}{\Rightarrow} \{Q\}$
alors $\{P\}$ **répéter** E **jusqu'à** B $\{Q \textit{ et } B\}$

Exemple : min d'une suite

```
{Antécédent : n>0}
{Conséquent : MinSuite(n) minimum d'une suite de n entiers lue
              sur l'entrée standard}
fonction MinSuite(donnée n: naturel) : entier
variables i type [0,n]
          x type entier
   min ← +∞
   i ← 0
   répéter
          i ← i+1
          lire(x)
          si x < min alors min ← x finsi
          {∀k ∈ [1,i], min⩽x_k}
   jusqu'à i = n
   {∀k ∈ [1,n], min⩽x_k et i=n}
   rendre min
finfonc {MinSuite}
```

- *Finitude* : $f(i) = n - i$

Énoncés itératifs en C

- l'énoncé **tantque** s'écrit :

```
/* {P} */ while (B) E /* {P et non B} */
```

- l'énoncé **répéter** s'écrit :

```
/* {P} */ do E while (B); /* {Q et non B} */
```

Exemple C : factorielle, calcul croissant

```
/* Antécédent : n>=0 */
/* Conséquent : factorielle(n) = n! */
unsigned long factorielle(const unsigned long n) {
  unsigned long i=0, fact=1;
  //Invariant : fact = i!
  while (i<n) {
    //fact = i! et i<n
    i++;
    fact*=i;
  }
  //i = n et fact = i! = n!
  return fact;
}
```

```
printf("%ld\n", factorielle(15)); //1 307 674 368 000
```

Exemple C : min d'une suite

```
#include <limits.h>
/* Antécédent : n>0 */
/* Conséquent : MinSuite(n) minimum d'une suite de n entiers lue
 *              sur l'entrée standard */
int MinSuite(const int n) {
  int i=0, min = INT_MAX;
  do {
      i++;
      int x;
      scanf("%d", &x);
      if (x<min) min=x;
      //∀k ∈ [1,i], min<=x_k
  } while (i!=n);
  //∀k ∈ [1,n], min<=x_k et i=n
  return min;
}
```

```
printf("min = %d\n", MinSuite(5));
```

Les énoncés pour

- On utilise ces énoncés lorsque le nombre d'itérations est connu à *l'avance*, c'est-à-dire de façon statique.
- Ils garantissent la *finitude* de la boucle (*i.e.* pas de condition de fin de boucle à programmer).

Forme générale

```
{P} pourtout x de T faire E finpour {Q}
```

Exemple

```
{écrire tous les carrés de 1 à n}
pourtout i de [1;n] faire
    écrire(i*i)
finpour
```

L'énoncé pour de C

- ne garantit pas l'achèvement de la boucle
- le programameur doit s'assurer de la finitude de la boucle

```
for (exp1 ; exp2; exp3) E
```

- équivalent à :

```
expr1;              //initialisation
while (exp2) {      //condition
  E;
  exp3;             //incrémentation
}
```

Exemples

```
/*
 * Antécédent : n>=0
 * Conséquent : factorielle(n) = n!
 */
unsigned long factorielle(const unsigned long n) {
  unsigned long fact=1;
  //Invariant : fact = i!
  for (unsigned long i=1; i<=n; i++)
    //fact = i! et i<=n
    fact*=i;
  //i>n et fact = n!
  return fact;
}
```