Les structures de contrôle et instructions composées

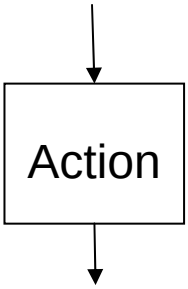

Fig. 4.1

Les algorithmes comportent des étapes ou des action. Chaque action à un début et une fin. Parce que l'algorithme est fini il peut être présenté comme une action ou un ensemble d'actions. Il y a quelques simples algorithmes qui s'appellent structures de contrôle, à l'aide de lesquelles on peut présenter un algorithme séquentiel de n'importe quelle complexité. Chaque structure peut être présenté comme une action avec une entrée et une sortie. Chacune d'eux peut contenir d'autres structures imbriquées et de telle façon de présenter l'algorithme avec différents degrés de détail (ig.4.1).

Chaîne (ou suite)

Dans la chaîne toutes les actions sont exécutées une après l'autre. Chaque action commence après la fin de la précédente (Fig.4.2). Dans le langage C ce sont des instruction ou des traduction des action écrits une après l'autre et encadrées an accolades (fig. 4.3). Les instructions encadrées en accolades

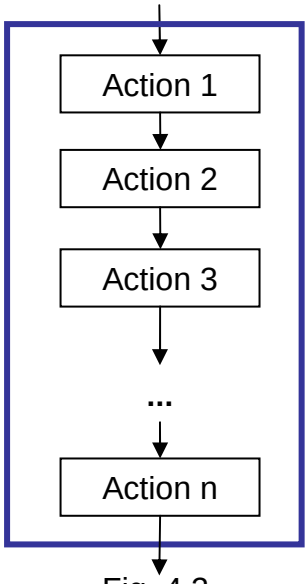

Fig. 4.2

```
{
  Instruction 1
  Instruction 2
  Instruction 3
  ...
  Instruction n
}
```

Fig. 4.3

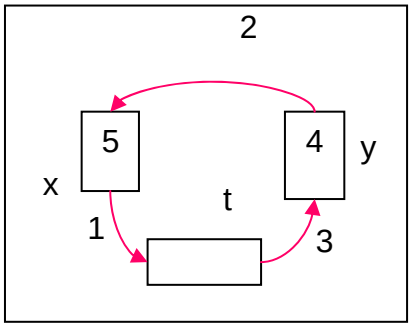

Fig. 4.4

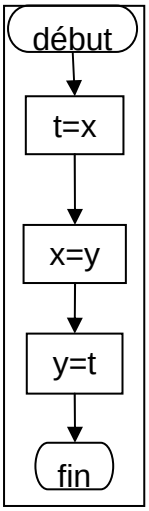

Fig. 4.5

```
#include <stdio.h>
void main(void){
  int x,y,t;
  /*initialisation de x et
y*/
  t = x;
  x = y;
  y = t;
}
```

Fig. 4.6

forment un block et sont considérées comme une instruction.

Exemple

Echanger les valeurs de deux variables entiers. Pour cette opération on a besoin d'une variable temporelle, dans laquelle on va stocker la valeur de la première variable pour ne pas la perdre (fig. 4.4). L'algorithme est montré à fig. 4.5 et le programme à fig.4.6.

Alternative

Alternative générale

L'alternative permet de brancher l'exécution du programme selon une condition. La condition est exprimée par une expression logique dont le résultat en C est 1 (vrai) ou 0 (faux). Une des action est exécutée mais jamais les deux à la fois. (fig. 4.7). La traduction en C par l'instruction **if** est montrée à Fig.4.8.

Exemples

1. **Trouver le nombre maximal de deux nombres entiers.**
   L'algorithme est dessiné à Fig.4.9 et le programme correspondant est écrit à fig. 4.10.
2. **Echanger le contenu de la variable *c* avec le contenu de cette variable (*a* ou *b*) qui a plus grande valeur.**
   L'algorithme est dessiné à Fig.4.11 et le programme correspondant est écrit à fig. 4.12. Notez bien comment les chaînes sont imbriquées dans l'alternative.

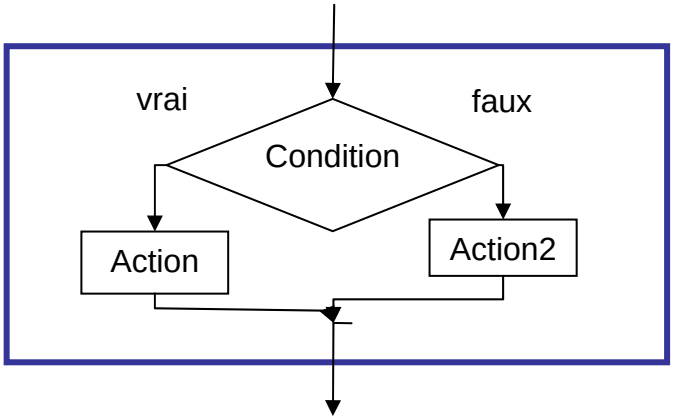

Fig. 4.7

```
if (condition 1)
   Traduction de Action 1
  else
   Traduction de Action 2
```

Fig. 4.8

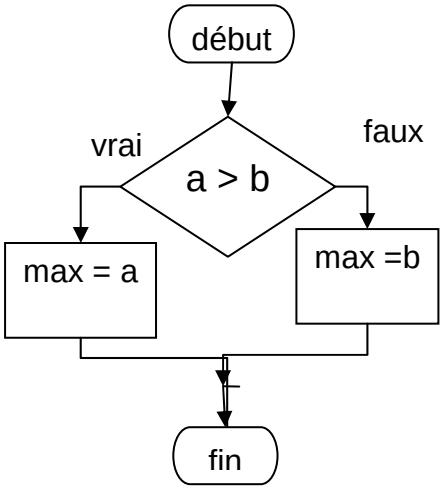

Fig. 4.9

```
#include <stdio.h>
void main(void){
  int a,b,max;
  /*initialisation de a et b*/
  if (a > b) max = a;
  else max = b;
}
```

Fig. 4.10

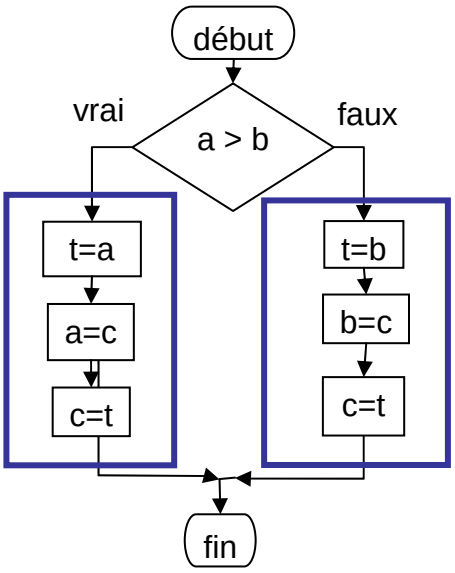

Fig. 4.11

```
#include <stdio.h>
void main(void){
  int a,b,max;
  /*initialisation de a, b et c */
  if (a > b) {
     t = a; a = c; c = t;
  }else {
     t = b; b = c; c = t;
  }
}
```

Fig. 4.12

.

Alternative simple

Quand une des branches d'une alternative (en général c'est la branche correspondante à la valeur faux) ne contient rien on dit que c'est une alternative simple (fig. 4.13 et 4.14). Dans l'instruction on peut omettre la partie **else**.

Exemple

Changer la valeur de la variable *a* avec sa valeur absolue.

La solution est donnée aux figures 4.15 et 4.16.

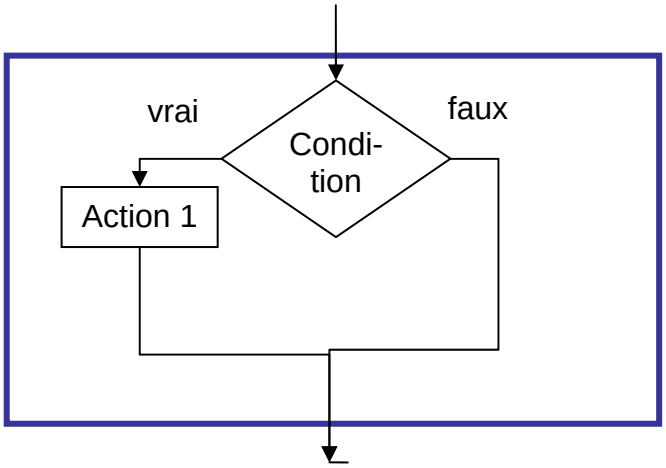

Fig. 4.13

```
#include <stdio.h>
void main(void){
  int a;
  /*initialisation de a */
  if (a < 0) a=-a;
}
```

Fig. 4.16

```
if (condition)
   traduction de
     Action 1
```

Fig. 4.14

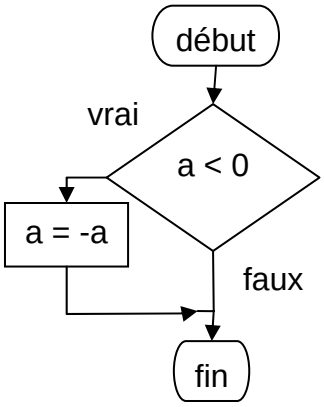

Fig. 4.15

Répétitive (ou boucle ou itération)

Il y a beaucoup de cas quand on a besoin de répéter une action plusieurs fois. Le cas le plus simple est quand on compte des objets. A chaque nouveau objet on ajoute un à un compteur dont la valeur initiale est zéro. Comme tous les algorithmes la répétition doit être finie. Pour cette raison on a besoin d'une condition pour terminer la répétition. Selon la place de la condition on peut distinguer des boucle avec précondition (« tant que »), boucles avec postcondition ou boucles mixte (la condition et le branchement sont au milieu du corps).

Boucle avec précondition (« tant que »)

L'organigramme de l'algorithme est dessiné à Fig. 4.17. La condition est calculée avant la première exécution de l'action répétée qu'on appelle très souvent le corps de la boucle. Cette structure est exécute de la façon suivante :

La condition est calculée et si elle est fausse (0) la boucle termine. Sinon le corps est exécuté et puis on reprend le calcul de la condition.

La particularité générale de cette boucle est que si au début le résultat du calcul de la condition est faux (0), le corps ne sera jamais exécuté. Cette boucle est traduit par l'instruction **while** (Fig.18).

Exemple

Compter les nombres entiers lus jusq'au on lit -100. On peut voir l'algorithme à Fig. 4.19 Noter qu'on doit avant la boucle de lire le premier nombre. Les premières deux action initialisent la boucle. L'action de lire à la fin du corps prépare la répétition suivante. L'initialisation de la boucle et la préparation de la répétition suivante sont présentes presque toujours dans les algorithmes répétitives. Le programme se trouve à Fig. 4.20.

Boucle for

Cette boucle est une cas particulier de « tant que », quand on a sait en avant le nombre de répétitions. Mais en C cette boucle et une boucle universelle de la boucle avec précondition où l'initialisation, la condition et la préparation sont mis dans l'instruction. L'instruction et l'organigramme correspondant sont montrés aux Fig. 4.21 et Fig. 4.22. Expr.1 c'est l'initialisation, expr2 – la condition et expr2 – la préparation. De telle façon la boucle de l'exemple précédent peut être écrit comme :

```
for (c=0, scanf("%d",n); n !=-100; scanf("%d",n)) c++;
```

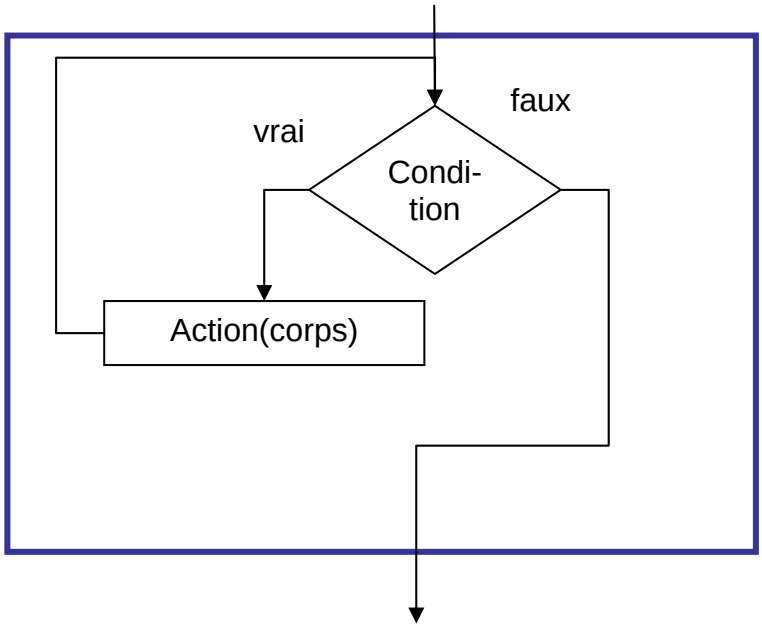

Fig. 4.17

```
while (condition)
   traduction de Action
```

Fig. 4.18

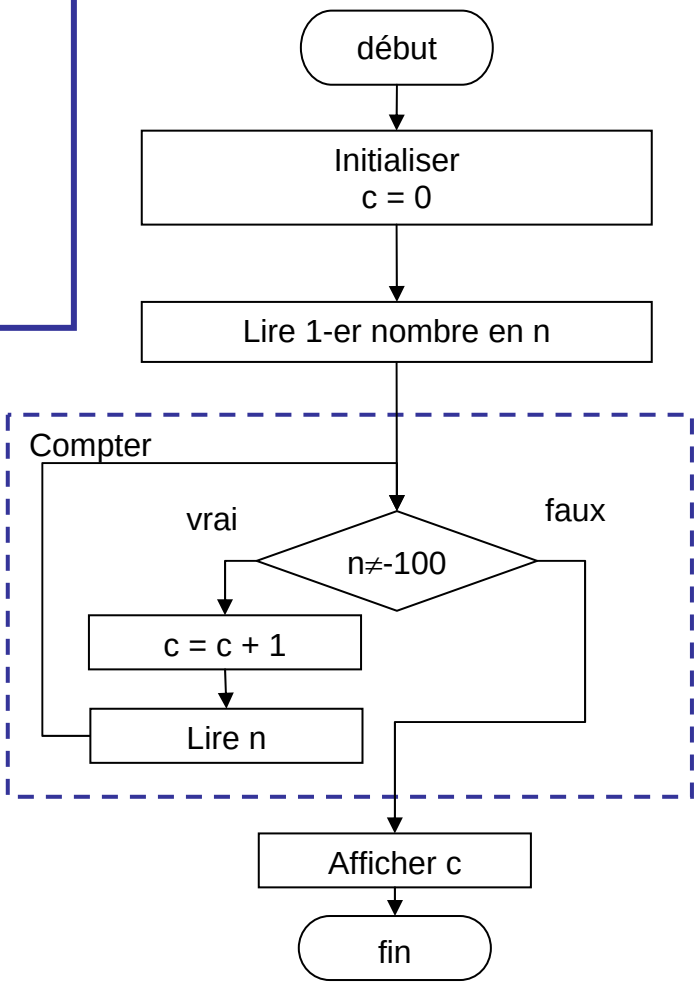

Fig. 4.19

```
#include <stdio.h>
void main(void){
  int n,c;
  c = 0; //initialisation
  scanf ("%d",&n);
  while (n !=-100) {
       c++;
       scanf ("%d",&n);
  }
  printf("le nombre est : %d\n",c);
}
```

Fig. 4.20

```
for (expr1;expr2;expr3)
   instruction
```

Fig. 4.21

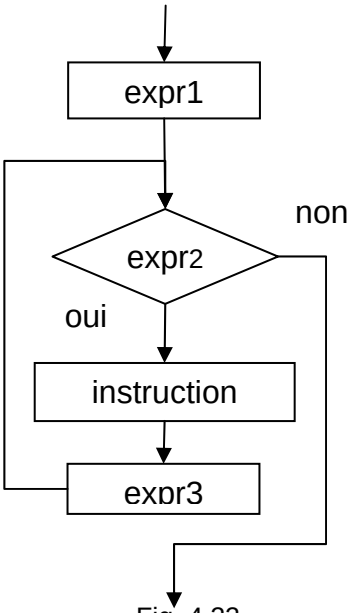

Fig. 4.22

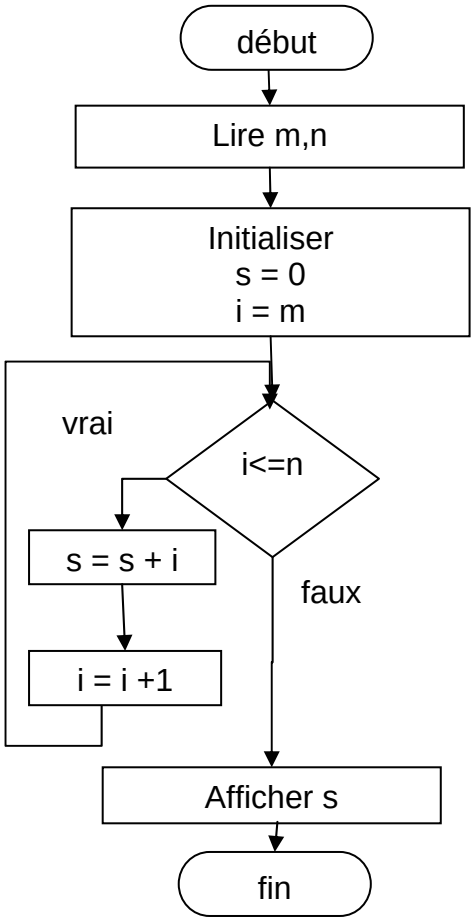

Fig. 4.23

```
#include <stdio.h>
void main(void){
  int m,n,s,i;
  printf("Tapez deux nombres:");
  scanf("%d %d", &m,&n);
  s = 0;
  i= m;
  while (i<=n) {
       s += i; i++;
  }
  printf("la somme est : %d\n", s);
}
```

Fig. 4.24a

```
#include <stdio.h>
#include <stdio.h>
void main(void){
  int m,n,s=0,i;
  printf("Tapez deux nombres:");
  scanf("%d %d", &m,&n);
  for (i= m;i<=n;i++) {
       s += i;
  }  printf("la somme est : %d\n", s);
}
```

Fig. 4.24b

Example

Sommer tous les entiers dans l'intervalle [M,N]. L'algorithme est à fig. 4.23 et deux programmes en C – la première avec une boucle **while** et la deuxième – avec une boucle for sont présentés aux fig. 4.24a et 4.24b.

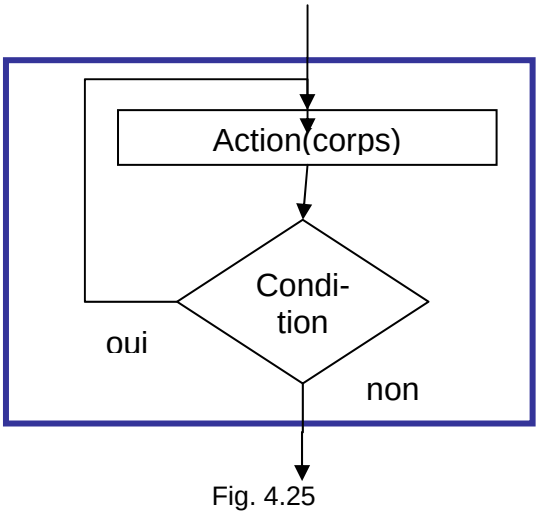

Fig. 4.25

```
do{traduction de
     Action(condition)
} while (condition)
```

Fig. 4.26

Boucle avec postcondition

Dans ce cas la condition est calculée après l'exécution du corps. Donc le corps sera exécuté au moins une fois. Cette boucle est nécessaire quand on ne peut calculer la condition au début parce que il manque certaines valeurs. L'organigramme est dessinée à Fig. 4.25 et le traduction en C – à Fig. 4.26.

Exemples

1. **Calculer la racine carrée d,un nombre réel *C* avec exactitude ε itérativement à l'aide de la formule de Héronнe :** $x_{i+1} = \frac{1}{2}\left(x_i + \frac{C}{x_i}\right)$ **où x_i est l'approximation connue et x_{i+1} est l'approximation suivante. On continue jusqu'à la valeur absolue de la différence entre deux approximations consécutives devient inférieure à ε.** L'algorithme est à Fig. 4.27 et le programme – à Fig.4.28.
2. **Faire la saisie d'un nombre qui doit être dans un intervalle.** C'est impossible de faire le test du nombre avant de le lire. le programme est à Fig.4.29. L'instruction "*if (y!=1) getchar();*" est écrite pour le cas quand un caractère invalide est tapée et la lecture est interrompue à ce caractère qui reste dans le tampon. Le résultat de scanf reste zéro et à la répétition suivante le scanf essaie de lire le même caractère avec le même résultat qui produit une boucle infinie,

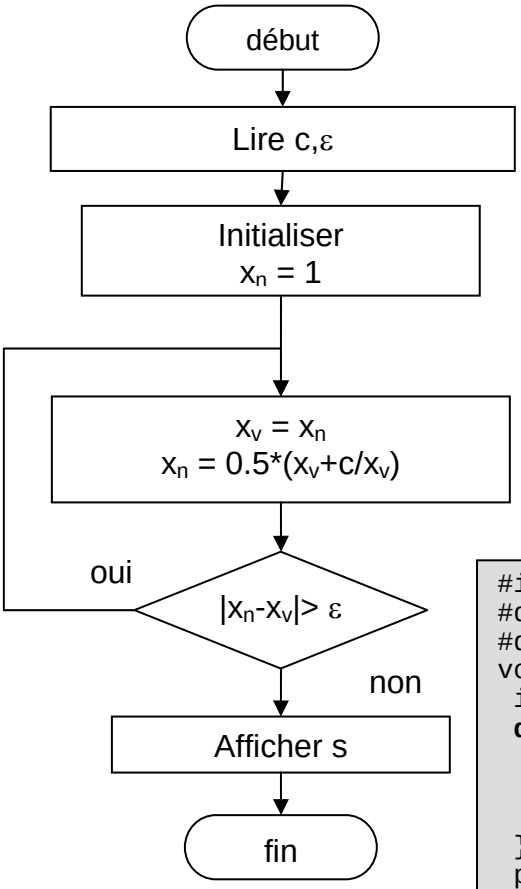

Fig. 4.27

```
#include <stdio.h>
#include <math.h>
void main(void){
  double c,xv,xn=1,eps;
  scanf("%lf %lf", &c,&eps);
  do {
       xv = xn;
       xn = 0.5*(xv+c/xv);
  }while (fabs(xn-xv)>eps);
  printf("le résultat est %9.3f\n",xn);
}
```

Fig. 4.28

```
#include <stdio.h>
#define  M 10
#define  N 20
void main(void){
 int x,y;
 do {
  printf ("Entrez un nombre entre %d et %d: ",M,N);
  y = scanf("%d",&x);
  if (y!=1) getchar();
 }while (y == EOF || y == 0 || x < M || x > N);
 printf ("\nLa valeur entrée est %d\n",x);
}
```

Fig. 4.29

Les instructions *break* et *continue*

Dans la portée d'une structure de contrôle (instructions **for**, **while**, **do** et **switch**), l'instruction break provoque l'abandon de la structure et le passage à l'instruction écrite immédiatement derrière. L'utilisation de break ailleurs qu'à l'intérieur d'une de ces quatre instructions constitue une erreur (que le compilateur signale). Dans l'exemple du compteur on ajoute la condition de ne pas compter après le quinzième nombre. L'algorithme et le programme sont montrés aux Fig. 4.30 et 4.31.

L'instruction continue est moins souvent utilisée. Dans la portée d'une structure de contrôle de type boucle (**while**, **do** ou **for**), elle produit l'abandon de l'itération courante et, le cas échéant, le démarrage de l'itération suivante. Elle agit comme si les instructions qui se trouvent entre l'instruction continue et la fin du corps de la boucle avaient été supprimés *pour l'itération en cours* : l'exécution continue par le test de la boucle (précédé, dans le cas de for, par l'exécution de

l'expression de préparation). Lorsque plusieurs boucles sont imbriquées, c'est la plus profonde qui est concernée par les instructions break et continue. Dans l'exemple de la somme dans un intervalle est ajoutée la condition de ne pas sommer les nombre qui se divisent exactement à 5. Le programme est montré à Fig. 4.32.

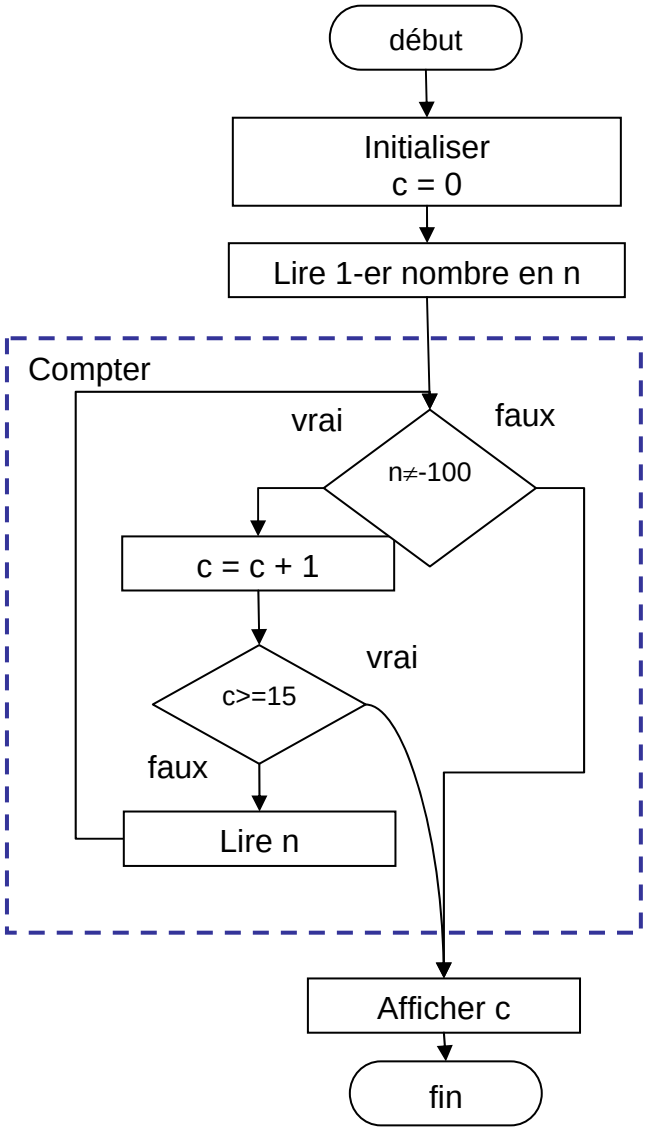

Fig. 4.30

```
#include <stdio.h>
void main(void){
  int n,c;
  c = 0; //initialisation
  scanf ("%d",&n);
  while (n !=-100) {
       c++;
        if (c>=15) break;
       scanf ("%d",&n);
  }
  printf("le nombre est : %d\n",c);
}
```

Fig. 4.31

```
#include <stdio.h>
#include <stdio.h>
void main(void){
  int m,n,s=0,i;
  printf("Tapez deux nombres:");
  scanf("%d %d", &m,&n);
  for (i= m;i<=n;i++) {
       if (i % 5 == 0) continue;
       s += i;
  }
 printf("la somme est : %d\n", s);
}
```

Fig. 4.32

La boucle mixte

Cette boucle nécessite la condition au milieu du corps (Fig. 4.33). Elle peut être exprimée par une boucle **while** infinie (la condition rend toujours vrai) et la vrai condition pour terminer incorporée dans un instruction **if** dont l'action est un **break** (Fig. 4.34). Une alternative de cette approche est montrée à Fig.4.35 – on copie la deuxième partie du corps avant la boucle comme initialisation.

Le Choix

Le choix est une structure dérivée de l'alternative. Il y a un ensemble de conditions incompatibles (il n'y a qu'une d'eux qui peut être vraie dans un moment) et à chaque condition correspond une action on a la structure montrée à Fig. 4.36 et le programme à Fig.4.37. En C il y a une instruction spéciale pour le cas quand il y a une expression de type entier ou compatible avec entier dont la valeur doit être égale aux constantes différentes qui forment les cas différents (Fig. 4.38). C'est l'instruction **switch** (Fig. 4.39). Ici on doit noter les instructions **break** qui se trouvent après chaque cas. Si ces instructions sont omis tous les instructions qui suivent le cas choisi seront exécutées m^me qu'elles n'appartiennent pas à ce cas.

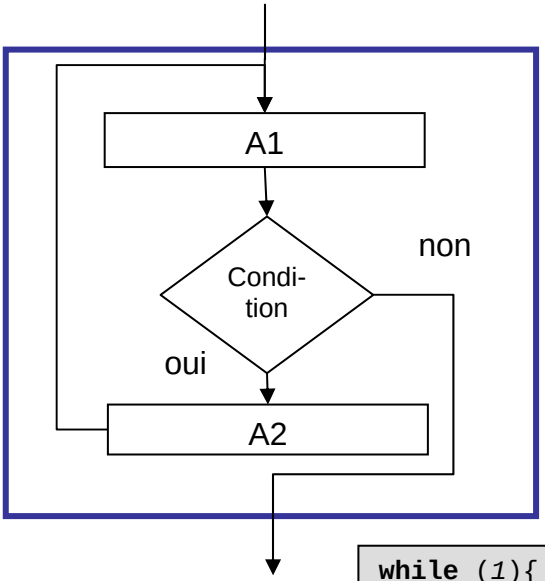

Fig. 4.33

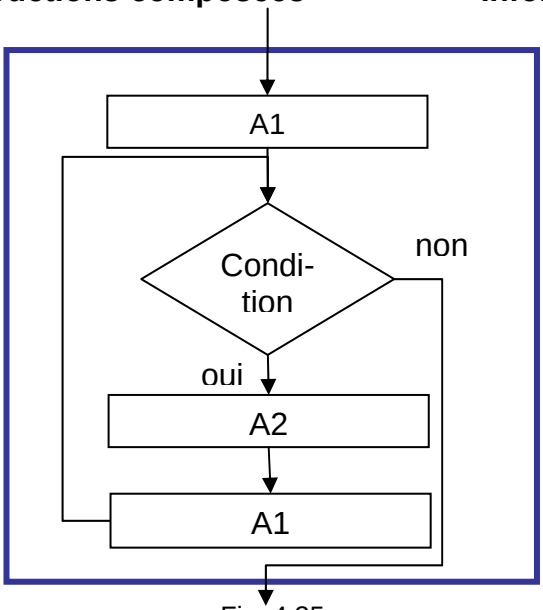

Fig. 4.35

```
while (1){
 A1;
 if (cond) break;
 A2;
}
```

Fig. 4.34

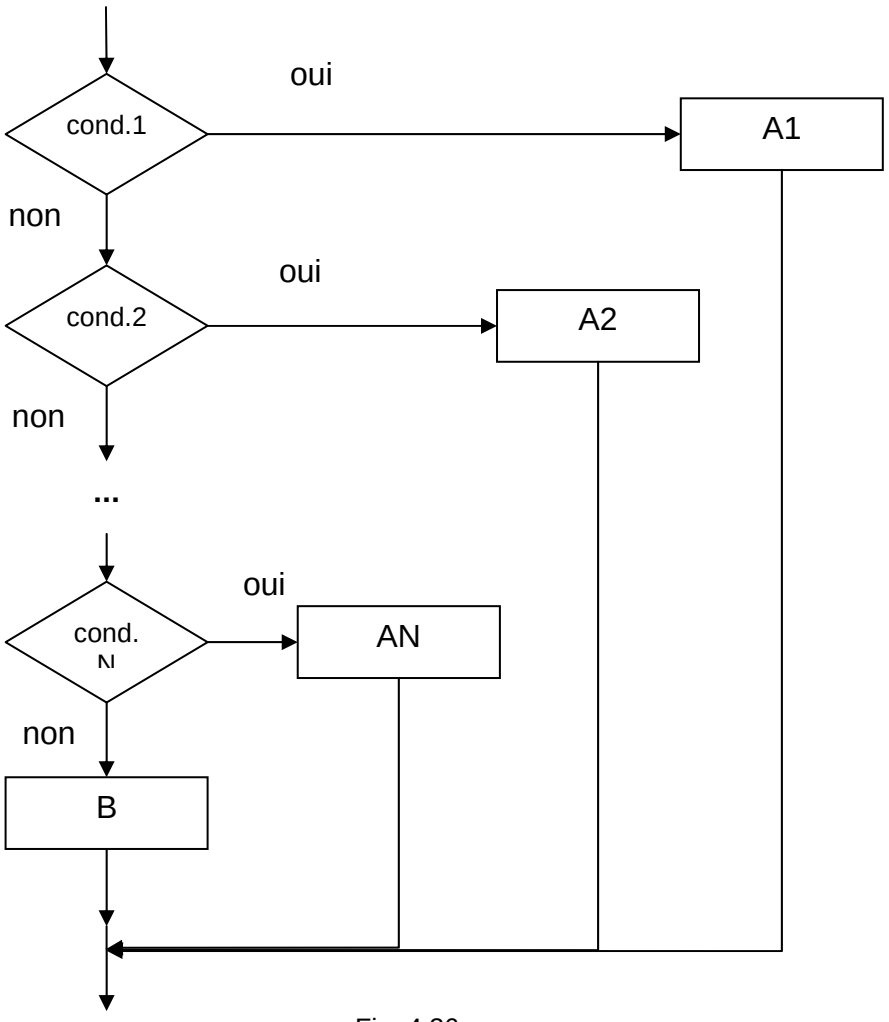

Fig. 4.36

```
if (cond.1)
   A1
  else if (cond.2)
        A2
   else if ...
    ...
     else if (cond.N)
        AN
      else
        B ;
```

Fig. 4.37

Exemple

Compter les chiffres, les blanc (espaces et tabulations), les points et tous les autres caractères dans un texte qui est tapé au clavier. L'algorithme et le programme sont montrés au Fig. 4.40 et 4.41. Notez bien que ici on a uni l'action

de saisie et le test pour la fin de la boucle. C'est possible en C parce que l'affectation est une expression et produit une valeur.

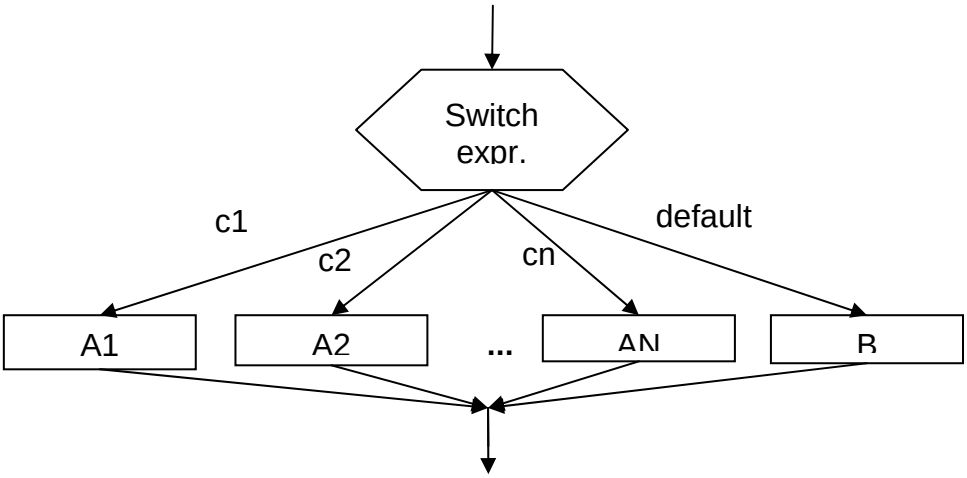

Fig. 4.38

```
switch (expr.){
  case c1: A1; break;
  case c2: A2; break;
  ...
  case cn : AN; break;
  default: B;
}
```

Fig. 4.39

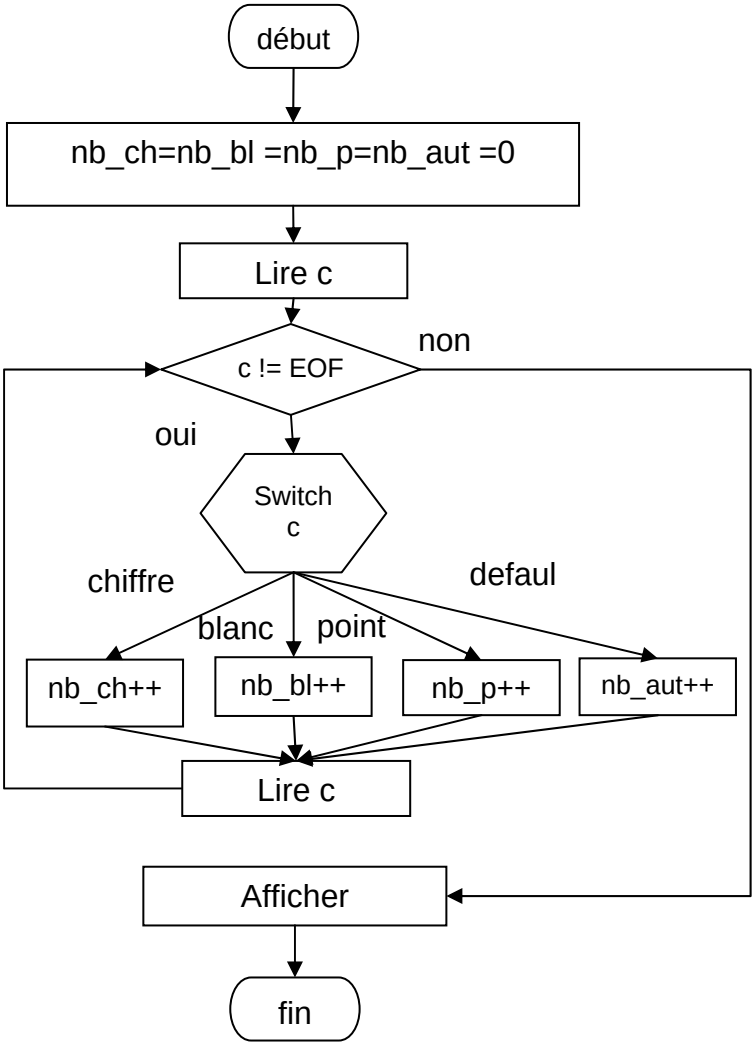

Fig. 4.40

```
#include <stdio.h>
void main(void){
int nb_ch, nb_bl, nb_p, nb_aut,c;
  nb_ch= nb_bl= nb_p=nb_aut=0;
  while ((c=getchar())!=EOF){
    switch(c){
      case '0':
      case '1':
      case '2':
      ...
      case '9': nb_ch++;break;
      case ' ':
      case '\t':nb_bl++;break;
      case '.': nb_p++; break;
      default : nb_aut++;
    }
  }
  printf("\n%d %d %d %d \n",
      nb_ch,nb_bl,nb_p,nb_aut);
}
```

Fig. 4.41