Initiation au C
cours n°3

Antoine Miné

École normale supérieure

1er mars 2007

Plan du cours

- un peu plus sur les boucles : `for`, `do`, `break`,
- les tableaux,
- les constantes symboliques.

Les boucles

Boucles `while` (rappels)

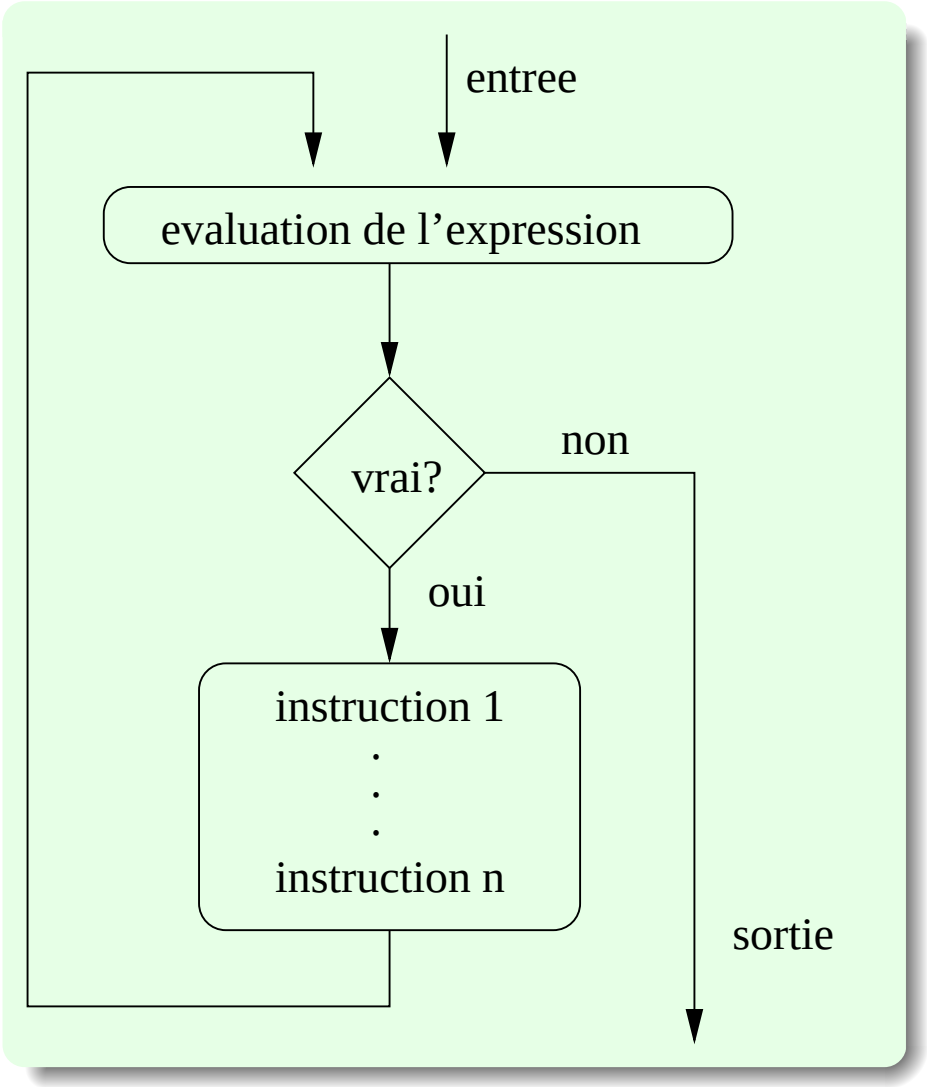
Syntaxe

```
while (expression) { instructions }
```

Effet :

- tant que *expression* est vraie, le bloc est exécuté,
- si la condition est initialement fausse,
  le bloc n'est jamais exécuté,
- la condition est re-testée après chaque "tour" de boucle.

Déroulement d'une boucle `while` (rappels)



Boucles do while

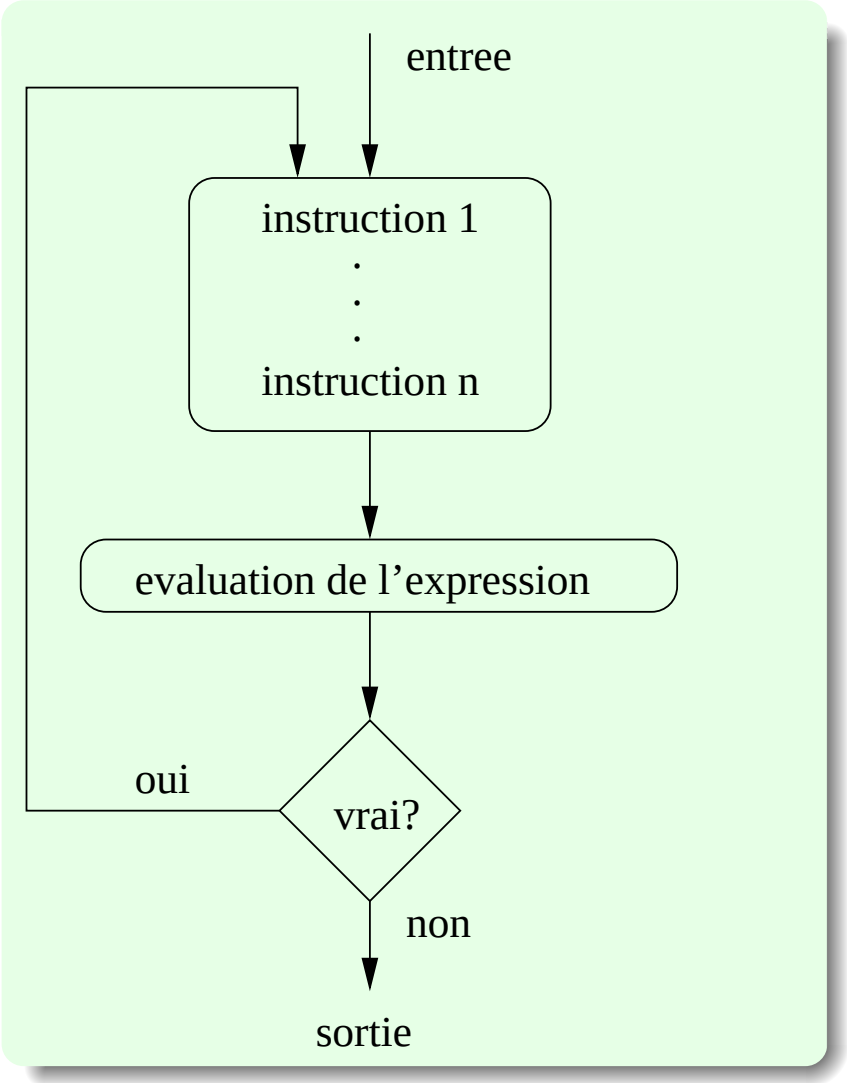
Syntaxe

```
do { instructions } while (expression) ;
```

Effet :

- le bloc est exécuté tant que *expression* est vraie,
- le bloc est toujours exécuté au moins une fois, même si la condition est initialement fausse,
- la condition est testée après chaque "tour" de boucle.

Déroulement d'une boucle do while



Boucles for

Syntaxe

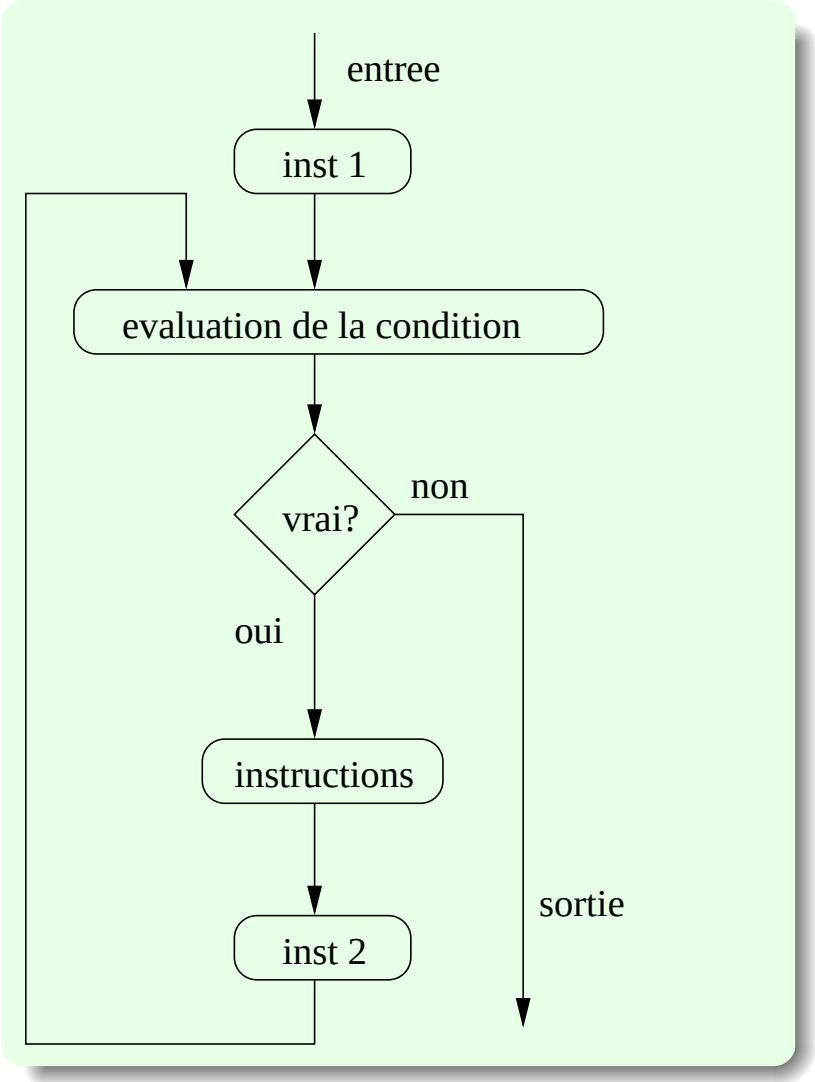
```
for (inst1 ; condition ; inst2)  { instructions }
```

- *inst1* et *inst2* : instructions simples,
- *condition* : expression booléenne,
- *instructions* : bloc d'instructions.

Effet : "boucle `while` avec initialisation"

- *inst1* est d'abord exécuté, une seule fois,
- tant que *condition* est vraie, exécute *bloc* puis *inst2*,
- la *condition* est testée :
  - après *inst1*,
  - après *inst2* à la fin de chaque "tour" de boucle.

Déroulement d'une boucle `for`



Exemples de boucle for

compte.c

```
int i;
for (i=0;i<15;i++) printf("%i ",i);
```

Résultat : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

compte_a_rebours.c

```
int i;
for (i=5;i>=0;i--) printf("%i ",i);
```

Résultat : 5 4 3 2 1 0

Notes syntaxiques

inst1 et **inst2** :

- peuvent être vides ⇒ pas d'action,
- peuvent contenir plusieurs instructions, séparées par des virgules **,**

condition :

- peut être vide ⇒ boucle infinie.

Extension C99 et C++ :

- **inst1** peut contenir une déclaration de variables

```
for (int i=0 ;i<10 ;i++)
```

elles sont détruites en sortie de boucle.

Exemples avancés de boucles for

boucle_infinie.c

```
for (;;) printf("Je boucle\n");
```

table_ASCII.c

```
int col,lin,c;
for ( lin=0, c=32; lin<6; lin++ ) {
  for ( col=0; col<16; col++, c++ )
    printf("%c ",c);
  printf("\n");
}
```

L'instruction break

break sort immédiatement de la boucle la plus imbriquée (for, do ou while).

Exemple

```
for (x=2;x<y;x++) {
  if ( ! (y%x) ) {
    printf("%i divise %i\n",x,y);
    break;
  }
}
/* on arrive ici après break */
```

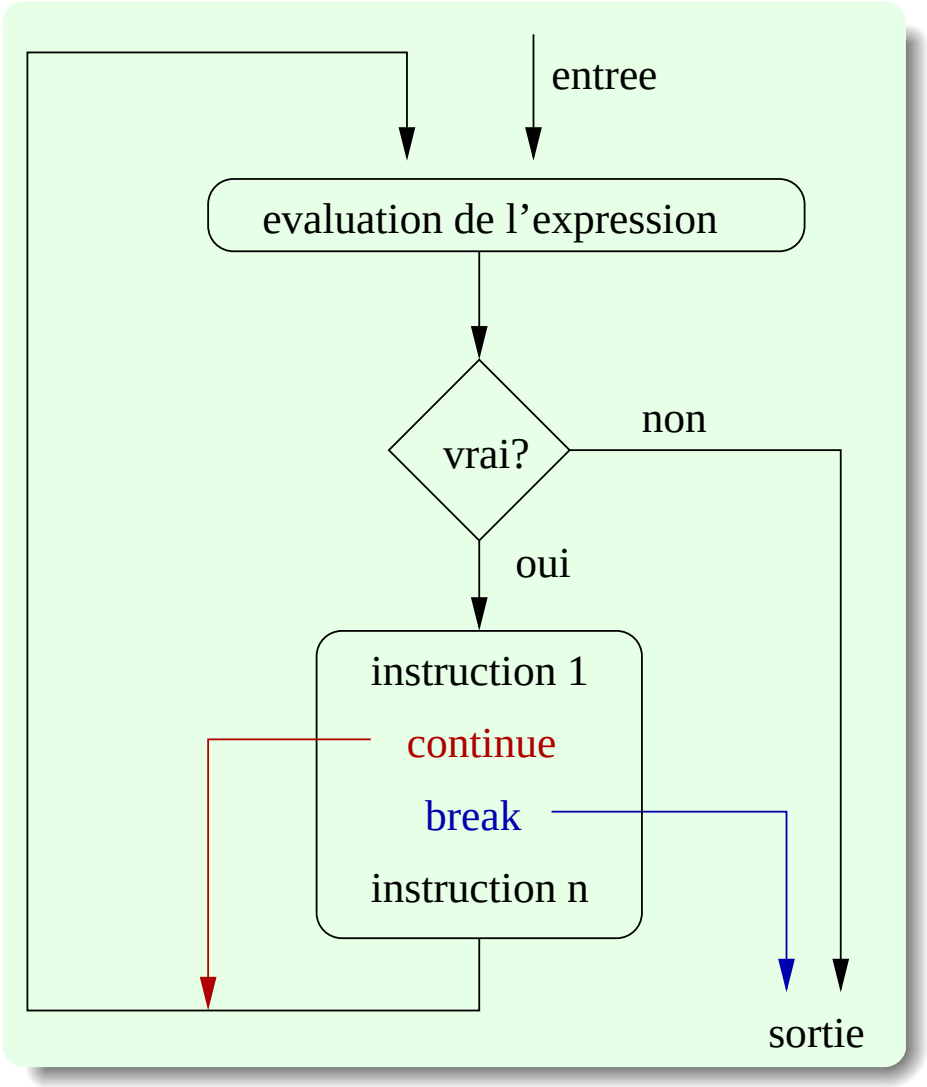
L'instruction continue

`continue` saute à la prochaine itération de la boucle.
La condition d'arrêt est testée.

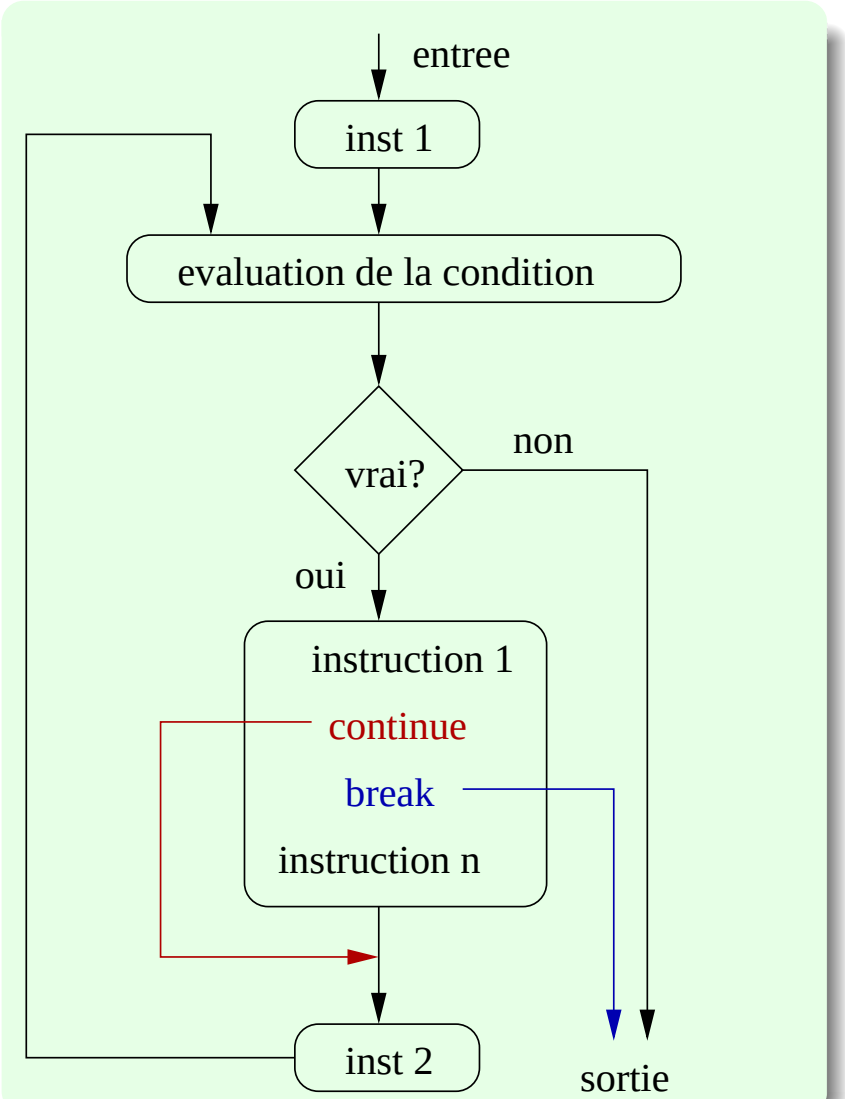
Exemple

```
double x, y=10.0;
for (x=0;x<100;x++) {
  if (x==y) continue;
  printf("%f\n",1.0/(x-y));
}
```

Effet sur une boucle `while`



Effet sur une boucle for



Les tableaux

Déclarations de tableaux

Tableau = séquence de 'cases mémoires' de même type.

Déclaration d'une variable–tableau

```
type var[taille] ;
```

Exemples de déclaration

```
int    tab[10];                 /* 10 entiers */
double Dalmatiens[10*10+1];     /* 101 flottants */
```

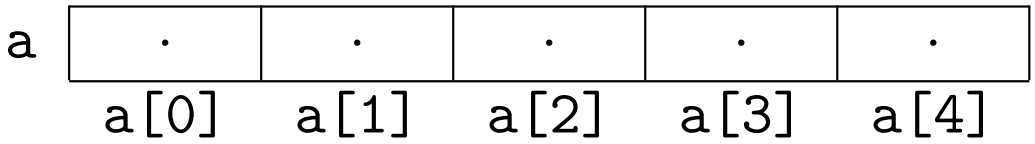
Taille :

- en C ANSI, `taille` doit être une expression constante *i.e.*, qui ne fait pas intervenir de variable,
- cette limitation est supprimée en C99 pour les variables locales.

Accès dans un tableau

Chaque case contient une valeur indépendante.
Les cases sont indicées de 0 à `taille-1`.

Exemple : `int a[5] ;`



Accès dans un tableau

`tab[expr]`

où

- `tab` est une variable de type tableau,
- `expr` est une expression entière.

Accès dans un tableau

`tab[expr]` est modifiable !

Lvalue = expression qui peut se trouver à gauche de = :

- une variable ‘scalaire’,
- un accès dans une variable–tableau.

Exemple

```
int a[10];
for (i=0; i<10; i++) a[i] = i*(i+1);
for (i=0; i<10; i++) printf("%i ",a[9-i]);
```

Par contre, une variable–tableau n’est pas une lvalue. . .

Accès dans un tableau

Puissance des tableaux : on peut accéder à un indice *calculé* par une expression quelconque.

Attention

Ne jamais accéder à un tableau en dehors de ses bornes !

- le langage C ne vérifie pas les accès dans les tableaux !
- le résultat d'un accès invalide est imprévisible :
  - peut planter le programme immédiatement avec `Segmentation fault`
  - peut ne rien faire de grave,
  - peut corrompre silencieusement la mémoire.

Initialisation de tableaux

Avant affectation, le contenu du tableau est aléatoire.

Déclaration avec initialisation

```
type var[taille] = { expr1, ..., exprN };
```

- les expressions sont séparées par des virgules `,`
- les accolades `{ }` et le point-virgule `;` sont obligatoires.

Notes :

- on doit avoir `N`≤`taille`,
- si 1≤`N`<`taille`, les initialiseurs manquants sont mis à 0,
- si `taille` est omise, elle est fixée à `N`.

Initialisation de tableaux

Exemples corrects

```
int a[3];                /* non initialise */
int a[3] = { 2,1,0 }; /* complètement initialise */
int a[3] = { 2,1 };    /* équivalent au précédent */
int a[]  = { 2,1,0 }; /* équivalent au précédent */
```

Exemples incorrects

```
int a[2] = { 2,1,0 }; /* trop d'initialiseurs */
int a[];                 /* taille inconnue */
a = { 1,2,3 };           /* pas une déclaration */
```

Copies de tableaux

Attention

On ne peut pas affecter un tableau entier.

Ne pas faire...

```
int a[10], b[10];
a = b; /* erreur de syntaxe! */
```

...mais plutôt une boucle

```
int i;
for (i=0;i<10;i) a[i] = b[i];
```

Il faut connaître à priori la taille du tableau,

`sizeof(a) / sizeof(a[0])`

Tableaux multidimensionnels

Tableau multidimensionnel = tableau de tableaux :

- matrices,
- images *bitmap*, etc.

Déclaration

```
type var[taille1][taille2]···[tailleN];
```

Pas de limite au nombre de dimensions.

Seule limitation : la taille mémoire

$$\texttt{sizeof(var)} = \texttt{sizeof(type)} \times \texttt{taille1} \times \cdots \times \texttt{tailleN}$$

e.g. : `double cube[64][64][64]` occupe 2 Mo.

Accès dans un tableaux multidimensionnel

Accès

`var[expr1][expr2]···[exprN]`

Attention :

- il faut autant d'expressions qu'il y a de dimensions,
- chaque `exprI` doit être compris entre 0 et `tailleI-1`.

Exemple : matrice identité

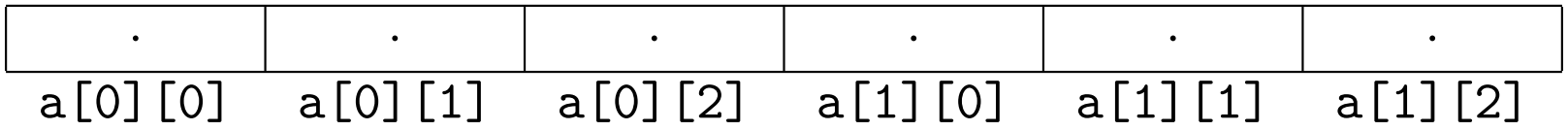
```
double mat[4][4];
int i,j;
for (i=0;i<4;i++)
  for (j=0;j<4;j++)
    mat[i][j] = (i==j) ? 1. : 0.;
```

Représentation en mémoire

Représentation en mémoire :

- `taille1` tableaux de `taille2` tableaux de ...
  de tableaux de `tailleN` éléments de type `type`.
- les éléments se succèdent en mémoire, sans 'trous'.

Exemple : `int a[2][3] ;`



$\Rightarrow$ `int a[2][3] ;` $\simeq$ `int a[2*3] ;`

`a[i][j]` $\simeq$ `a[(i)*3+(j)]`

Initialisation des tableaux multidimensionnels

Déclaration avec initialisation : { et } imbriqués.

Exemple

```
int a[2][3] = { { 1,0,1 }, { 0,1,0 } };
```

Copie de tableaux : par des boucles imbriquées.

Passage de tableaux en paramètres

Attention

Les tableaux sont passés par référence, pas par valeur !

- pas de mémoire allouée pour l'argument,
- pas de copie du tableau passé en paramètre,
- un accès à l'argument modifie le tableau passé en paramètre.

Attention

Une fonction ne peut pas retourner de tableau !

Exemple

Exemple de passage par référence

```
void zero(int a[4]);

int main()
{
  int tmp[4];
  zero(tmp);
  return 1;
}

void zero(int a[4])
{
  int i;
  for (i=0;i<4;i++) a[i] = 0; /* modifie tmp */
}
```

Les constantes symboliques

Définition de constantes symboliques

Syntaxe (baroque)

```
#define CST valeur
```

- `CST` : identificateur, par convention en majuscules,
- `valeur` : texte arbitraire,
- doit occuper une ligne complète,
- pas de point-virgule `;` final.

Effet : dans la suite du programme, `CST` est remplacé par `valeur`.

Application des constantes symboliques

Application :
écrire du code *paramétrique* là ou le C interdit les variables.

Exemple

```
#define N 10
int x[N], y[N];
void f()
{
  int i;
  for (i=0;i<N;i++)
    y[i] = x[i] + 1;
}
```

Pour changer la taille de tous les tableaux,
il suffit de changer une seule ligne.

Danger des constantes symboliques

Définition de constante symbolique $\neq$ affectation de variable !

- affectation : évaluation,
- constante symbolique : substitution littérale.

$\Rightarrow$ danger de "capture" syntaxique.

Exemple d'erreur

```
#define N x+y
z = 3*N; /* signifie z = 3*x+y, pas z = 3*(x+y) */
         /* x et y peuvent aussi symboliques! */
```

Correction

```
#define N ((x)+(y))     /* plus sûr */
```