



UNIVERSITÉ
CÔTE D'AZUR

Préprocesseur

Présentation: Stéphane Lavirotte

Auteurs: ... et al*



**(*) Cours réalisé grâce aux documents de :
Erick Gallezio**

Mail: Stephane.Lavirotte@univ-cotedazur.fr

Web: <http://stephane.lavirotte.com/>

Université Côte d'Azur

Préprocesseur: fonctions de base

- ✓ Le préprocesseur C est appelé **avant** la compilation du fichier source
- ✓ Les directives du préprocesseur sont sur une ligne commençant par le caractère #
- ✓ Principales fonctions:
 - Substitutions textuelles (`#define`)
 - définition de constantes
 - définition de macros
 - inclusion de texte (`#include`)
 - compilation conditionnelle (`#if`, `#ifdef`, `#ifndef`)



Substitutions: définition de constante

✓ Syntaxe:

```
#define identificateur chaîne
```

✓ Exemples:

```
#define FALSE 0
#define TRUE 1
#define EOF (-1)

#define SIZE 1024
#define SIZE2 (2 * 1024)
int buf[SIZE], big_buf[SIZE2];

#define IF if (
#define THEN ){
#define ELSE ;} else {
#define ENDIF ;}

...
IF a < b THEN x = y + z; z = w ELSE a = 2 * b ENDIF
/* équivalent à */
if (a < b) { x = y + z; z = w;} else {a = 2 * b;}
```



Substitutions: définition de constante 1/2

✓ Syntaxe:

```
#define identificateur(x1, x2, ...xn) chaîne
```

✓ Cela permet de définir des pseudo fonctions:

- plus rapide (puisque pas de coût d'appel et de retour de fonction)
- souvent plus lisible que la macro expansion

✓ Exemples:

```
#define getchar() getc(stdin)
```

```
#define putchar(c) putc(c, stdout)
```

```
#define min(a, b) ((a) < (b) ? (a) : (b))
```

```
#define max(a, b) ((a) > (b) ? (a) : (b))
```

```
while ((c = getchar()) != EOF)
```

```
m = min(m, c);
```

```
==> while ((c = getc(stdin)) != (-1))
```

```
m = ((m) < (c) ? (m) : (c));
```

Substitutions: définition de constante 2/2

✓ Quelques dangers des macros

– La substitution est seulement textuelle

```
#define bad_max(a, b) a > b ? a : b
x = 2 + bad_max(x, y);
==> x = 2 + x > y ? x : y;
x = (2 + x) > y ? x : y;
```

– Attention aux effets de bord

```
x = max(a--, b--);
==> x = ((a--) > (b--) ? (a--) : (b--)); /* max
décrémenté 2 fois */
```

– Attention aux performances

```
x = max(a[i*3 + 2*j][k+1], b[2*i]);
==> x = ((a[i*3 + 2*j][k+1]) > (b[2*i]) ?
(a[i*3 + 2*j][k+1]) : (b[2*i]));
/* Les index sont calculés plusieurs fois */
```



Substitutions: stringification

✓ ANSI C définit un opérateur de *stringification*

```
#define TRACE(x) printf("%s = %d\n", #x, x)
TRACE(3*2+4);
==> printf("%s = %d\n", "3*2+4" , 3*2+4);
==> 3*2+4=10 // affiché sur stdout
```

✓ Cette macro pourrait aussi être écrite comme:

```
#define TRACE(x) printf(#x " = %d\n", x)

TRACE(3*2+4);
==> printf(#x " = %d\n", 3*2+4);
==> printf("3*2+4" " = %d\n", 3*2+4);
==> printf("3*2+4 = %d\n", 3*2+4);
==> 3*2+4=10 // affiché sur stdout
```



Substitutions: concaténation dans les macros

✓ **ANSI C définit un opérateur de concaténation ##**

```
#define Positif(fct) \  
    int Positif_##fct(double x) { \  
        double res = fct(x); \  
        return res < 0 ? 0 : res; \  
    }  
...  
Positif(sin);  
Positif(cos);  
...  
==> int Positif_sin(double x)  
    {  
        double res = sin(x);  
        return res < 0 ? 0 : res;  
    };  
int Positif_cos(double x)  
    {  
        double res = cos(x);  
        return res < 0 ? 0 : res;  
    };
```

Substitutions: définition de macro à la compilation

- ✓ Il est possible de définir une macro
 - sans modifier le code source
 - à la compilation

- ✓ Exemple:

```
gcc -c -DMAXBUF=150 -DOS=linux -DDEBUG buffer.c
```

- ✓ est équivalent à

```
#define MAXBUF 150  
#define OS linux  
#define DEBUG
```


Substitutions: macro prédéfinies

✓ Les macros suivantes sont prédéfinies:

- `__LINE__`: le numéro de la ligne actuelle
- `__FILE__`: le nom du fichier actuel
- `__DATE__`: la date de la compilation
- `__TIME__`: l'heure de la compilation
- `__STDC__`: 1 si le compilateur est conforme à la norme ANSI
- `__STDC_VERSION__`: vaut 199901L si C99



Substitutions: oublier une définition

- ✓ **L'oubli d'une définition (macro ou constante) se fait avec `#undef`**

```
#undef IF
```

```
#undef THEN
```

```
#undef ELSE
```

```
#undef ENDIF
```

Substitutions: évaluation du mécanisme

✓ Avantages

- permet de modifier la syntaxe du langage
- permet de définir des fonctions *inline*

✓ Inconvénients

- permet de modifier la syntaxe du langage
- permet de définir des fonctions *inline*



Inclusion: inclusion de fichier source

✓ Deux formes:

```
#include <fichier>
```

- **cherche dans la liste de répertoires standard ("/usr/include" sur Unix)**

```
#include "fichier"
```

- **cherche dans le chemin spécifié et, si absent, dans la liste de répertoires standard.**

✓ Exemples:

```
#include <stdio.h>
```

```
#include <X11/X11.h>
```

```
#include "test.h"
```

```
#include "../..../prog.h"
```



Inclusion: ajout de chemins standards

✓ L'option `-I` du compilateur

- permet de rajouter des chemins à la liste de chemins d'inclusion standard
- ordre important

✓ Example:

```
$ gcc -I../../my-include -I/usr/local/include foo.c
```

✓ Lors de l'inclusion du fichier `<incl.h>`, on cherche

```
../../my-include/incl.h  
/usr/local/include/incl.h  
/usr/include/incl.h
```

Compilation conditionnelle

1/5

✓ Ce mécanisme permet de:

- paramétrer des structures de données complexes à la compilation
- éviter la compilation de code inutile () gain mémoire)
- prendre des décisions à la compilation plutôt qu'à l'exécution, quand c'est possible (=> gain de temps).

✓ Syntaxe:

```
#if                #if
    lignes_1        lignes_1
#else              #endif
    lignes_2
#endif
```

✓ Il y a trois formes de test:

```
#if expression statique
#ifdef identificateur
#ifndef identificateur
```

Compilation conditionnelle

2/5

✓ Exemples:

```
#ifndef MAXSIZE
    #define MAXSIZE 1024
#endif
typedef struct {
    #if LINUX || MACOS
        int _cnt; unsigned char *_ptr;
    #else
        unsigned char *_ptr; int cnt;
    #endif
    unsigned char *_base; ...
} FILE;

#if MAX > 1024
    #error MAX must be less than 1024
#else
    int buffer[MAX];
#endif
```

Compilation conditionnelle

3/5

✓ Exemples (suite):

```
#ifndef DEBUG
    #define trace(s) printf(s)
#else
    #define trace(s)
#endif

trace("calling f()");
/* mais trace("calling g(%f, %d)", x, i) est impossible */

/* éviter les re-définitions en cas d'inclusion multiple */
#ifndef INCL_H
    #define INCL_H
    définitions de incl.h
#endif
```


Compilation conditionnelle

4/5

✓ Pour définir des macros avec un nombre variable de paramètres:

✓ **C ANSI**

```
#ifndef DEBUG
    #define trace(s) printf s
#else
    #define trace(s)
#endif

trace(("calling f()")); /* ==> printf ("calling f()"); */
trace(("calling g(%f, %d)", x, i)); /* ==> printf ("calling g(%f, %d)", x, i); */
```

✓ **C99**

```
#ifndef DEBUG
    #define trace(...) printf(__VA_ARGS__)
#else
    #define trace(s)
#endif

trace("calling f()"); /* printf("calling f()"); */
trace("calling g(%f, %d)", x, i); /* printf("calling g(%f, %d)", x, i); */
```

Compilation conditionnelle

5/5

✓ Exemples (suite):

```
#ifdef DEBUG
#define assert(cond) \
    if (!(cond)) { \
        fprintf(stderr, \
            "assertion " #cond " failed in file %s, \
            line %d\n", __FILE__, __LINE__); \
        abort(); \
    }
#else
#define assert(cond)
#endif

/* Utilisation */
assert(i < MAX && t[i] > b);
==>
assertion i < MAX && t[i] > b failed in file foo.c line 42
```



Exécution du préprocesseur

- ✓ L'option `-E` du compilateur C, permet de lancer seulement le préprocesseur
 - affichage du résultat sur la sortie standard

- ✓ On peut donc utiliser le préprocesseur C
 - pour des langages qui n'ont pas de préprocesseur
 - pour faire des substitutions textuelles ou des inclusions