Préprocesseur C

Définition

Le **préprocesseur** est un programme qui applique certaines transformations au code **avant la compilation**.

Commandes

Les commandes préprocesseurs **commencent** toutes par le signe **# en début de ligne**.

Inclusion

```
#include "someCodeFile"
```

L'inclusion copie le contenu d'un fichier texte spécifié à l'emplacement de la commande.

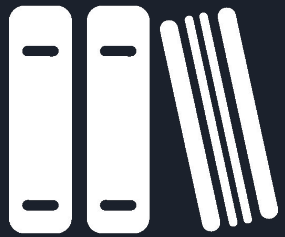
L'inclusion est utilisée pour :

- **Mutualiser** du code commun à plusieurs fichiers.
- Rendre le code **accessible** à un code externe.

Inclusion

L'inclusion joue un rôle central dans la programmation de **classes et de bibliothèques**.

Pour une classe, elle contient la **déclaration** de ses **variables et méthodes.**



Pour une bibliothèque, elle contient la **déclaration** de son **API**.

define

```
#define PI 3.14159265358979323846
#define RAWTEXT Some text here
```

`#define` permet de déclarer un **identifiant de précompilation** associé à du **texte brut**.

define

```
#define EQUAL_PI = 3.1415
float someVar EQUAL_PI;
```

Lorsque cet **identifiant** est utilisé par ailleurs dans le code, il est **remplacé par le texte associé** avant compilation.

```
float someVar = 3.1415;
```

define

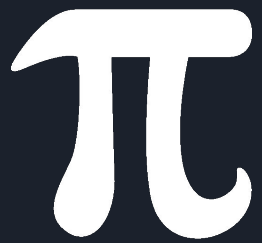
```
#define DEBUG
```

`#define` peut également être déclaré sans texte associé.
Dans ce cas, il sera utilisé avec `#ifdef` et `#ifndef`.

Utiliser un identifiant défini sans texte dans le code ne changera rien à ce dernier.

define

Define est utilisé pour :



Créer des **constantes en pré-compilation**

Créer des **synonymes** pour **simplifier/accélérer** la programmation

Utiliser les commandes `#ifdef` et `#ifndef`.

ifdef

```
#ifdef DEBUG
...
#endif // DEBUG
```

Le compilateur ne prendra en compte le code entre `#ifdef` et `#endif` que si l'identifiant associé **a été défini** en amont dans le **fichier source**.

ifndef

```
#ifndef DEBUG
...
#endif // DEBUG
```

Le compilateur **ignorera** le code entre `#ifndef` et `#endif` si l'identifiant associé **a été défini** en amont dans le **fichier source**.

If

```
#if expression
 …
#endif // expression
```

Le compilateur ne prendra en compte le code entre #if et #endif que si **l'expression associé est vraie**.

If

```
#define VERSION 5
#if VERSION < 4
#endif
```

L'expression peut contenir :

- Des identifiants de define.
- Le mot-clé defined(...).
- Des opérateurs tels que +, ==, &&, !, <, ...
- Des valeurs 4, true, "text", ...

If

```
#ifdef DEBUG
```

Équivaut à

```
#if defined(DEBUG)
```

Le mot-clé `defined` dans une expression vérifie si l'identifiant **est bien défini en amont**.

Elif else

```
#if VERSION > 4
#elif VERSION == 3
#elif !defined(VERSION)
#else
#endif
```

`#elif` et `#else` marchent comme une structure `if`, `elseif`, `else` pour le préprocesseur.

If/ifdef/ifndef/elif/else/endif

`#if`, `#ifdef`, `#ifndef`, `#elif`, `#else` et `#endif` permettent de sélectionner le **code à compiler** en **fonction d'une configuration** : debug, release, versionning, ...

La configuration projet **peut déclarer des defines.**

Dans les inclusions, ces commandes permettent également de **vérifier qu'un code n'existe pas déjà**.

Macros

```
#define ADD(x,y) x + y
int i = ADD(1,2);
```

Il est possible d'ajouter des paramètres à l'identifiant d'un `#define` que l'on appel alors **macro**.

Lors de l'utilisation de la **macro**, **les paramètres sont recopiés** dans la texte associé au `#define`.

Macros

```
#define MULT(x,y) x * y
int i = MULT(1+2,3+4);
```

Équivaut à

```
int i = 1+2*3+4;
```

Attention, les paramètres sont recopiés **en texte brut** sans être évalués au préalable !

Macros

```
#define CONCAT(x,y) x##y##1
int i = CONCAT(3,2);
```

Équivaut à

```
int i = 321;
```

La concaténation **##** permet d'effectuer une **fusion** de paramètres dans une **macro**.

Macros

```
#define TO_STR(text) #text
string i = TO_STR(Some text);
```

Équivaut à

```
string i = "Some text";
```

Le symbole **#** dans une **macro** permet de **changer** le paramètre en **chaîne de caractères**.

Macros

```
#define TO_STR(text) #text
int i = 0;
printf(TO_STR(i));
```

Équivaut à

```
printf("i"); // Not "0"
```

Attention, les paramètres sont copiés en **texte brut**.

Macros

Les macros sont utilisées pour **créer des synonymes** et **simplifier** la programmation.

```
#define IS_LOG_ACTIVE true
#define LOG(text) if(IS_LOG_ACTIVE)\
                      { printf(text); }
LOG("Some Text\n")
```

Pragma once

```
#pragma once
```

`#pragma once` empêche la double inclusion.

En en-tête d'un fichier destiné à être inclu, `#pragma once` assure que le fichier **ne sera inclu qu'une seule fois** par les fichiers l'incluant.