En plus de contenir les prototypes des fonctions de la bibliothèque standard du C, les fichiers d'en-têtes de celle-ci contiennent la définition d'un certain nombre de valeurs utiles. Par exemple, le fichier math.h contient une valeur approchée du nombre PI, très utilisé en mathématiques. Pour vous éviter d'avoir à écrire la valeur 3.1416, ou une valeur plus précise, à chaque fois que vous voulez utiliser PI dans un calcul, une *constante* est définie dans le fichier `math.h`, sous la forme suivante :

```
#define M_PI 3.14159265358979323846
```

Cette ligne permet de définir que dans la suite du programme, toutes les occurences de l'identifiant `M_PI` seront remplacées par le nombre se trouvant à sa droite. Par exemple, le programme suivant affiche le périmètre d'un cercle dont le rayon est tapé au clavier :

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

int main()
{
   double rayon;
   double perimetre;
   scanf("%lf", &rayon);
   perimetre = rayon * 2 * M_PI;
   printf("%lf\n", perimetre);
   return 0;
}
```

Les instructions `#include` et `#define`, ainsi que d'autres qui commencent également par le caractère '#', ne sont pas des instructions traitées par le compilateur C lui-même, pour être transformées en instruction du microprocesseur. Il s'agit en fait d'instructions destinées à un programme qui lit votre source et y effectue des modifications, avant de le transmettre au compilateur. Ce programme est appelé *préprocesseur* : il traite (processe) votre source avant (pré) de le transmettre au compilateur, après quelques modifications.

Par exemple, dans le source ci-dessus, le préprocesseur remplace les instructions #include par le contenu complet des fichiers spécifiés et remplace l'identifiant M_PI par sa valeur, avant que le compilateur lui-même ne commence son travail.

Vous pouvez bien sûr définir vous-mêmes vos propres constantes. Par exemple, si vous voulez définir un programme qui demande l'âge de l'utilisateur, puis lui indique si cette personne est majeure, vous pouvez définir une constante `AGE_MAJORITE` qui vaudra 18, puis l'utiliser dans votre code :

```
#include <stdio.h>

#define AGE_MAJORITE 18

int main()
{
   int age;
   scanf("%d", &age);
   if (age >= AGE_MAJORITE)
      printf("Vous êtes majeur\n");
   else
      printf("Vous êtes mineur\n");
   return 0;
}
```

Les identifiants pouvant être utilisés pour définir des constantes sont les mêmes que ceux permettant de définir des variables ou des fonctions. Pour éviter de les confondre, on décide par convention que les noms de constantes définies par un `#define` doivent être en majuscules.

L'instruction `#define` permet d'associer bien plus que de simples valeurs numériques à des identifiants. On peut en fait placer n'importe quel texte à droite de l'identifiant de la constante, après un espace. Le préprocesseur remplacera toutes les occurences de l'identifiant dans la suite du programme, par le texte situé à droite de l'identifiant, dans l'instruction `#define`.

Par exemple, on peut écrire :

```
#define AGE_MAJORITE age * 2 - 8
```

Dans ce cas, le programme sera modifié par le préprocesseur de la manière suivante :

```
if (age >= age * 2 - 8)
   printf("Vous êtes majeur\n");
else
   printf("Vous êtes mineur\n");
```

Il est possible de placer des instructions #define à n'importe-quel endroit du programme, à condition qu'elles se trouvent au début de la ligne. Si un même identifiant est utilisé pour plusieurs #define, le préprocesseur remplace les occurences de l'identifiant par la valeur définie lors du dernier #define correspondant rencontré.

Exercice : dans l'exemple suivant, faites à la main le travail du préprocesseur, en remplaçant toutes les occurences de constantes par leur valeur (sans remplacer les #include par le contenu du fichier). Déterminez ensuite (sans l'exécuter) ce que ce programme affichera.

```
#include <stdio.h>

#define PI 3.1416
#define PI_2 PI * 2
#define PI_BIS PI_2 - 3.1416

int main()
{
   printf("PI : %lf\n", PI);
   printf("PI_2 : %lf\n", PI_2);
   printf("PI_BIS : %lf\n", PI_BIS);
   printf("PI_BIS * 2 : %lf\n", PI_BIS * 2);
   return 0;
}
```

Solution : voici le code obtenu après remplacement par le préprocesseur (sauf #include) :

```
#include <stdio.h>

int main()
{
   printf("PI : %lf\n", 3.1416);
   printf("PI_2 : %lf\n", 3.1416 * 2);
   printf("PI_BIS : %lf\n", 3.1416 * 2 - 3.1416);
   printf("PI_BIS * 2 : %lf\n", 3.1416 * 2 - 3.1416 * 2);
   return 0;
}
```

On peut remarquer que lorsque le texte "PI" apparaît entre guillemets dans le source, il n'est pas remplacé par le préprocesseur : il n'est remplacé que lorsqu'il est utilisé en tant qu'identifiant, là où l'on pourrait placer une variable.

Voici maintenant le résultat de l'exécution de ce programme :

```
PI : 3.141600
PI_2 : 6.283200
PI_BIS : 3.141600
PI_BIS * 2 : 0.000000
```

L'affichage de la dernière ligne n'est pas très naturel : au lieu d'être le double de la ligne précédente, comme pourrait le suggérer le code `PI_BIS * 2`, c'est 0 qui est affiché. L'expression obtenue après remplacement est en effet `3.1416 * 2 - 3.1416 * 2`. Les priorités des opérateurs font alors que la multiplication par 2 ne se fait pas sur l'ensemble de ce que représente PI_BIS, mais uniquement sur le nombre qui suit le signe -. Pour éviter ce genre de surprises, lorsque l'on définit des constantes qui valent des expressions, on prend donc l'habitude de mettre des parenthèses autour de l'expression :

```
#define PI_BIS (PI_2 - 3.1416)
```
