Note : Codes sources disponibles uniquement en C.

Utiliser une fonction qui n'existe pas

Note : dans tous les autres langages du site, la déclaration implicite est impossible ; l'utilisation d'une fonction non déclarée produit simplement une erreur.

Essayez de compiler le programme suivant :

```
int main()
{
   test();
}
```

Vous obtiendrez un message de ce genre :

```
↳ test.c: In function `main':
test.c:3: warning: implicit declaration of function `test'
/tmp/cc1pfH2s.o: In function `main':
/tmp/cc1pfH2s.o(.text+0x7): undefined reference to `test'
collect2: ld returned 1 exit status
```

La première partie du message n'est pas une erreur, mais un avertissement (*warning*) vous indiquant qu'il y a un risque d'y avoir une erreur dans votre programme, mais que ce n'est pas certain. L'avertissement `implicit declaration of function `test'` indique que la fonction *test* a été déclarée implicitement. Cela signifie tout simplement que vous l'avez appelée sans qu'elle ait été déclarée.

Le compilateur C peut en effet accepter d'appeler une fonction non déclarée, en devinant son prototype à partir de l'appel. Par exemple avec l'appel `test()`, le compilateur considère que *test* est une fonction qui ne prend aucun argument. Il considère aussi systématiquement que le type de retour est `int` (il ne peut pas le deviner à partir de l'appel).

L'utilisation de déclarations *implicites*, si elle peut parfois fonctionner, est fortement déconseillée. En effet, si le type deviné par le compilateur n'est pas le bon, il se produira une erreur à l'exécution, puisque la fonction n'aura pas les paramètres qu'elle attend. Cet avertissement vous prévient donc qu'il y a un risque, même si ce que vous faites est autorisé.

La deuxième partie du message est par contre une erreur qui empêche la compilation de se terminer : la fonction *test* n'est pas trouvée dans votre code ni dans la bibliothèque standard et l'exécutable ne peut donc être généré.

Nombre de paramètres

Avec le programme suivant :

```
#include <stdio.h>

int somme(int nb1, int nb2)
{
   return nb2 + nb1;
}

int main()
{
   printf("%d\n", somme(42));
}
```

vous obtiendrez un message comme celui-ci :

```
↳ test.c: In function `main':
  test.c:10: too few arguments to function `somme'
```

Le message se traduit en français : « trop peu d'arguments pour la fonction *test* ». L'erreur vient du fait que vous avez appelé la fonction test avec un seul paramètre, alors que son prototype indique qu'il en faut deux. Si vous aviez au contraire mis 3 paramètres lors de l'appel, vous auriez obtenu l'erreur suivante :

```
↳ test.c:10: too many arguments to function `test'
```

Cette erreur se traduit en « trop de paramètres pour la fonction *test* ».

Valeur de retour

Dans certains langages (Java, JavaScool et OCaml sur le site), cette étourderie produit une erreur.

Avec le code suivant :

```c
#include <stdio.h>

int plusGrand(int a, int b)
{
   if (a > b)
      return a;
   if (a < b)
      return b;
}

int main()
{
   printf("%d\n", plusGrand(42, 43));
}
```

vous obtiendrez un message tel que :

```
↳ test.c: In function `plusGrand':
  test.c:9: warning: control reaches end of non-void function
```

Il s'agit ici d'un avertissement et non d'une erreur, ce qui signifie qu'un exécutable sera tout de même généré et que vous pourrez lancer votre programme. L'avertissement vous indique cependant qu'il y a de grandes chances pour que vous ayez commis une erreur à la ligne 9 de votre programme, c'est-à-dire à la fin de votre fonction *test*. Le message se traduit en « avertissement : le contrôle atteint la fin d'une fonction non `void` » (donc une fonction qui retourne une valeur).

L'avertissement indique qu'il est possible que l'exécution atteigne la fin de la fonction. Ceci ne devrait pas arriver avec une fonction qui doit retourner une valeur : une instruction `return` devrait systématiquement être exécutée avant la fin du bloc d'instructions. Dans la fonction *test*, il est en effet possible qu'aucun des deux tests ne soit vérifié donc qu'aucune des deux instructions `return` ne soit exécutée.

Erreurs peu explicites

Devinez d'où vient l'erreur dans le programme suivant :

```
```

```c
#include <stdio.h>
int somme(int nb1; int nb2)
{
   return nb1 + nb2;
}

int main()
{
   printf("%d\n", somme(42, 43));
}
```

le message étant :

```
↳ test.c:3: parameter `nb1' has just a forward declaration
```

L'erreur vient de la manière dont sont déclarés les paramètres de la fonction *test* : ceux-ci ne devraient pas être séparés par un point-virgule, mais par une virgule.

Le message d'erreur en lui-même est difficile à comprendre, même pour des programmeurs expérimentés. On peut le traduire par « le paramètre *nb1* n'est déclaré que de manière anticipée ». La « déclaration anticipée » permet de déclarer que des éléments existent avant qu'un espace mémoire leur soit effectivement alloué, un peu comme pour les prototypes de fonctions. Ce genre de choses est cependant très peu utilisé et ne correspond en tout cas pas à l'erreur réellement commise, qui est une simple faute de frappe.

Il peut arriver fréquemment que des messages d'erreur ou des avertissement soient difficiles à comprendre. Ils sont cependant toujours là pour une bonne raison : un compilateur n'invente jamais (ou très très rarement) des erreurs. Regardez donc le numéro de la ligne à laquelle elle se produit et relisez bien le code qui s'y trouve, ou le code concernant des éléments qui y sont utilisés.