École de technologie supérieure

ILLUSTRATION: De l'importance des prototypes

INF125 Introduction à la programmation
Sylvie Ratté et Hugues Saulnier

Ce document nécessite une lecture préalable des thèmes 1 et 2.

1. LE CAS DU PROTOTYPE ABSENT

Les prototypes de fonction fournissent des informations capitales au compilateur: le type de la valeur retourné, le nombre et le type de charque arguments nécessaires. Ces informations sont essentielles si l'on désire que le compilateur effectue correctement son travail. Leur absence entraîne parfois des résultats surprenants.

A. LA LIBRAIRIE OUBLIÉE (ou les particularités de Turbo C)

Nous savons désormais que la librairie "math" contient les prototypes de plusieurs fonctions utiles. La fonction "sqrt" (extraction de la racide carrée) en est un exemple. Son prototype, tel qu'il apparaît dans le fichier en-tête "math.h", se présente comme suit:

double sqrt(double x);

La fonction accepte donc un argument de type "double" et retourne, comme on peut s'y attendre, une valeur de type "double".

Pour utiliser cette fonction, nous savons aussi qu'il est nécessaire d'inclure la librairie "math" en fournissant la commande suivante au préprocesseur:

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

Qu'arrive-t-il si l'on oublie d'inclure cette librairie? Et bien le résultat semble dépendre grandement du compilateur utilisé. En Turbo C, vous pouvez saisir le petit programme suivant qui affiche simplement le résultat de la racine carrée de 100.

```
#include <stdio.h>
int main(){
   printf("%f\n", sqrt(100));
   return 0;
}
```

Avec votre intuition bien légitime, vous croyez que le programme affiche 10.0? Mais vous avez tort. Le programme affichera probablement 0.0 ou même une valeur négative! Aucune erreur n'est mentionnée par le compilateur ou l'éditeur de liens, pas même un petit avertissement. Mieux encore, si vous modifiez le programme comme suit:

```
#include <stdio.h>
int main(){
   printf("%f\n", sqrt(100, 5, 40));
   return 0;
}
```

Vous remarquez tout de suite que la fonction est appelée avec 3 arguments alors qu'elle en demande normalement 1 seul! Le compilateur Turbo C ne mentionne aucune erreur une fois de plus. Et le programme affiche toujours un mauvais résultat. Par contre, si vous ajoutez le prototype, le tour est joué et le compilateur puis l'éditeur de liens font correctement leur travail.

Nous n'avons pas tenté l'expérience avec d'autres fonctions. À vous de faire des tests.

B. LE PROTOTYPE PERSONNEL OUBLIÉ

Lorsque notre programme utilise une fonction qui nous appartient (ex. la fonction "estPremier"), le fait d'oublier la mention du prototype ne permet pas au compilateur de détecter les erreurs dans les énoncés d'appel. Ainsi, les deux programmes suivants peuvent être compilés sans erreur (évidemment l'éditeur de liens ne pourra pas générer le code exécutable). Vous remarquez que le second programme utilise la fonction avec 3 arguments.

```
#include <stdio.h>
int main(){
  int i;
  for (i = 3; i < 500; i++)
    if (estPremier(i) == 1)
      printf("%i est un nombre premier.\n", i);
  return 0;
}

#include <stdio.h>
int main(){
  int i;
  for (i = 3; i < 500; i++)
    if (estPremier(i, 5, 8) == 1)
      printf("%i est un nombre premier.\n", i);
```

```
    return 0;
}
```

Il est donc essentiel que vous mentionniez vos prototypes AVANT que le compilateur ne rencontre un énoncé d'appel à vos fonctions. C'est pourquoi vous retrouvez les prototypes AVANT la fonction "main". De plus, cela facilite la création d'une librairie éventuelle puisqu'il suffira, pour ce faire, de transférer les prototypes dans un fichier .H et les définitions dans un fichier .C.

2. LE CAS DU PROTOTYPE INCOMPLET

Le problème exposé à la section précédente concerne les prototypes absents et nous en avons conclu qu'il était nécessaire de toujours déclarer nos prototypes de fonction. Mais se pourrait-il que même en présence d'un prototype nous devrions être sur nos gardes?

Nous connaissons tous les fonctions "main" typiques d'un programme C et très souvent, nous l'écrivons comme suit:

```
int main(){...}
```

Les parenthèses vides indiquent, nous le savons, l'absence d'arguments. Cette absence peut aussi être explicitement mentionnée en utilisant le mot réservé "void" comme suit:

```
int main(void){...}
```

Qu'arrive-t-il si l'on déclare une fonction personnelle avec des parenthèses vides? Un bon exemple est donné par la petite fonction suivante:

```
double alea();
```

Cette fonction retourne un nombre aléatoire dans l'intervalle [0..1]. Il importe peu de connaître sa définition, contentons-nous de son prototype et utilisons-la dans un petit programme qui affiche 10 nombres aléatoires.

```
#include <stdio.h>
double alea();
int main(){
   int i;
   for (i = 0; i < 10; i++)
     printf("%i\n", alea());
   return 0;
}
```

Si l'on commande une compilation, tout se passe bien, comme on pouvait s'y attendre. Modifions quelque peu le programme:

```
#include <stdio.h>
double alea();
int main(){
   int i;
   for (i = 0; i < 10; i++)
     printf("%i\n", alea(45, 6));
   return 0;
}
```

Commandons une compilation. De manière tout à fait surprenante et inattendue (pour qui ne connaît pas les petits défauts du C ANSI), le compilateur n'offre aucune erreur, pas même un avertissement. Pourtant, il aurait raison de se plaindre puisque la fonction est appelée avec 2 argument alors qu'elle en exige aucun! Modifions le prototype pour rendre cette exigence explicite:

```
double alea(void);
```

Cette fois, le compilateur effectue correctement son travail et mentionne l'erreur.

Conclusion générale

En C ANSI, déclarez toujours vos prototypes de la manière la plus explicite possible.
