Saisie au clavier et lecture d'un flot de données

Fondements du langage C

La lecture d'un flot de données à partir d'un fichier, en général, ou la saisie de données en provenance du clavier, en particulier, sont des opérations fréquemment implémentées dans les logiciels. En langage C, ces opérations peuvent être programmées en faisant appel à différentes fonctions de la bibliothèque standard. Ces fonctions sont généralement définies dans le fichier d'entête `stdio.h`.

Fonctions `scanf()` et `fscanf()` pour les saisies formatées

La fonction `scanf()` considère un nombre variable de paramètres. Elle est utilisée pour la saisie de données formatées, qu'il s'agisse de nombres, de caractères ou de chaînes de caractères. Il s'agit ainsi d'une fonction variadique dont le prototype est le suivant :

```
int scanf(const char *format, ...);
```

La fonction `scanf()` retourne le nombre de variables affectées par la saisie en cas de réussite. En cas d'échec, la macro `EOF` (**e**nd **o**f **f**ile) est exécutée et la constante `-1` associé est retournée. Il est alors possible de vérifier le bon déroulement en évaluant la valeur de retour.

La fonction `scanf()` utilise les mêmes spécificateurs de format que la fonction `printf()`. Ils sont listés dans le tableau 1.

Spécificateurs	Signification
%d ou %i	Entier signé
%u	Entier non signé
%o	Entier exprimé en octal
%x ou %X	Entier exprimé en hexadécimal
%c	Caractère
%f	Réel
%e	Réel en notation scientifique
%s	Chaîne de caractères

Tab. 1 – *Liste des spécificateurs de format*

Le code source présenté ci-dessous montre un exemple d'exploitation de la fonction `scanf()` :

```
int nb = 0;
int res;

printf("Entrez un nombre entier : ");
res = scanf("%d", &nb);

if (res != EOF)
  printf("La saisie est %d.\n", nb);
else
  printf("Erreur de saisie !\n");
```

La fonction `scanf()` recopie la saisie de l'utilisateur sur le flux d'entrée standard `stdin`, qui correspond généralement au clavier, à une adresse mémoire automatiquement affectée lors de la déclaration de la variable concernée par le recueil de la valeur saisie. Par conséquent, il est nécessaire de spécifier qu'il

s'agit de l'adresse de la variable et non pas de la variable elle-même. Ceci est réalisé en plaçant le caractère `&` devant le nom de la variable. Dans le cas ci-dessus, la variable `res` est déclarée pour stocker le nombre entier saisi et son adresse est alors spécifiée par `&res`.

Lorsqu'il s'agit de recueillir et de stocker une chaîne de caractères, il est nécessaire d'utiliser une variable multidimensionnée correspondant à un tableau de caractères. Dans ce cas de figure, il est inutile de spécifier l'adresse de la variable par le caractère `&` puisque celle-ci est implicitement renseignée par le nom de la variable. Le code source suivant présente un cas d'utilisation de la fonction `scanf()` utilisée pour la saisie d'une chaîne de caractères :

```
char nom[25];

printf("Entrez votre nom : ");
scanf("%s", nom);
```

La variable multidimensionnée `nom` permet de collecter une chaîne de vingt-quatre caractères au maximum. Il faut effectivement considérer le caractère de fin de chaîne `'\0'`.

Bien que simple d'utilisation, la fonction `scanf()` peut néanmoins présenter des problèmes dans certains cas d'utilisation. En effet, la récupération d'une saisie est partielle si la donnée contient un espace, une tabulation ou une entrée (retour chariot) qui est alors considérée comme la fin de saisie. Le reste de la donnée qui n'est pas pris en compte est cependant conservé en mémoire et risque d'être traité lors de l'appel suivant à la fonction `scanf()`. Il convient donc de l'utiliser avec précautions.

La fonction `fscanf()` est très similaire à la fonction `scanf()`. Cependant, elle ne considère pas par défaut une lecture du flux standard d'entrée `stdin`. Il est alors nécessaire de préciser le fichier correspondant au flux pris en compte. Le prototype de la fonction `fscanf()` est le suivant :

```
int fscanf(FILE *stream,
           const char *format, ...)
```

Le code ci-dessous montre un exemple simplifié de l'utilisation de la fonction `fscanf()` pour la saisie d'un nombre entier à partir du flux standard d'entrée :

```
int nb;

printf("Entrez un nombre entier : ");
fscanf(stdin, "%d", &nb);
```

Dès lors qu'elle prend une variable de type `FILE` comme premier paramètre, la fonction `fscanf()` peut être plus généralement utilisée pour récupérer les données contenues dans un fichier.

La valeur entière retournée par la fonction `fscanf()` est identique à celle retournée par la fonction `scanf()`.

Fonctions `getc()`, `getchar()` et `fgetc()` pour la récupération de caractères

Lorsqu'il s'agit de ne saisir et de ne prendre en compte qu'un unique caractère, il est préférable d'utiliser la fonction `gets()`, la fonction `getchar()` ou la fonction `fgetc()`. Ces trois fonctions sont définies dans le fichier d'entête `stdio.h`.

Le prototype des trois fonctions est le suivant :

```
int getc(FILE *stream);
int getchar(void);
int fgetc(FILE *stream);
```

Les trois fonctions renvoient la valeur entière non signée sur huit bits (`char`) convertie en entier signé sur seize bits (`int`) et correspondant au caractère lu.

Un exemple simplifié de mise œuvre de la fonction `getc()` est donné par le code ci-dessous :

```
#include <stdio.h>

int main(void)
{
   char car;

   printf("Entrez un caractère : ");
   car = getc(stdin);
   printf("Caractère : %c\n", car);
   return 0;
}
```

Les fonctions `getc()` et `fgetc()` s'utilisent strictement de la même façon. Elles prennent en paramètre un fichier ouvert sur un flot de données d'entrée. Dans le cas le plus courant, il s'agit du flot standard d'entrée noté `stdin` (**st**an**d**art `in`put) et qui correspond généralement au clavier.

La fonction `getchar()` ne prend aucun paramètre et considère systématiquement le flot standard d'entrée `stdin` comme fichier d'entrée. Elle est ainsi strictement identique à `getc(stdin)`.

Le code suivant est une adaptation de l'exemple précédent :

```
#include <stdio.h>

int main(void)
{
    char car;

    printf("Entrez un caractère : ");
    car = getchar();
    printf("Caractère : %c\n", car);
    return 0;
}
```

Si une erreur se produit au moment de l'exécution, la valeur `-1` associée à `EOF` est renvoyée par les trois fonctions `getc()`, `getchar()` et `fgetc()`.

Fonctions `gets()` et `fgets()` pour la récupération de chaînes de caractères

Il existe également plusieurs fonctions standards du langage C qui permettent de récupérer une chaîne de caractères. L'un des plus utilisées est la fonction `gets()` dont le prototype est le suivant :

```
char *gets(char *s);
```

La fonction `gets()` lit successivement les caractères d'une ligne à partir du flot de données d'entrée et stocke le résultat de la lecture dans la chaîne de caractères passée en paramètre. La lecture s'arrête lorsqu'un retour à la ligne (`'\n'`) est rencontré ou que la fin de fichier (`EOF`) est atteinte. De plus, la fonction `gets()` ajoute un caractère `NULL` à la chaîne de caractères lue dans le but d'en marquer la fin. Ce caractère est `'\0'`.

La fonction `gets()` présente un inconvénient majeur. En effet, si la taille du tableau destiné à recueillir la chaîne lue est insuffisante, le résultat est indéterminé. Par conséquent, l'utilisation de cette fonction n'est pas toujours recommandée.

Il est préférable de faire appel à la fonction `fgets()` dont le prototype est présenté ci-dessous :

```
char *fgets(char *s,
            int n,
            FILE *stream);
```

La fonction `fgets()` lit un flot de données à partir d'un fichier ouvert en lecture et qu'elle stocke dans un tableau pointé par le premier paramètre et noté `s` dans le prototype. Tout comme la fonction `gets()`, la lecture s'interrompt dès lors qu'un retour à la ligne (`'\n'`) ou que la fin de fichier est atteint `EOF`. Afin de marquer la fin de chaîne, le caractère `NULL` (`'\0'`) est ajouté après le dernier caractère lu.

Un exemple d'utilisation de la fonction `fgets()` est présenté ci-dessous :

```
#include <stdio.h>

int main(void)
{
  char nom[10];

  printf("Quel est votre nom ? ");
  fgets(nom, 10, stdin);
  printf("Votre nom est %s !\n", nom);

  return 0;
}
```

Le code présenté consiste à lire une chaîne de caractères transmise par le flot de données standard (`stdin`), qui correspond généralement au clavier, et à la stocker dans un tableau d'une taille maximale de dix caractères pour ensuite l'afficher.

La fonction `fgets()` retourne le même pointeur que celui pointé par l'argument `s` si la fonction s'est exécutée sans erreur. Elle retourne la valeur `NULL` si une erreur s'est produite. L'évaluation de la valeur retournée permet donc de savoir si l'exécution s'est correctement déroulée ou non.

Vidage du tampon d'entrée à l'aide de la fonction `fflush`

Il est assez courant de devoir s'assurer le tampon d'entrée est bien vide préalablement à une saisie. Cette opération s'effectue à l'aide de la fonction `fflush` dont le prototype est le suivant :

```
int fflush(FILE *stream);
```

Un exemple classique et fréquent d'utilisation de la fonction `fflush` est présenté dans la code simplifié ci-dessous :

```
fflush(stdin);
```

Plus largement, la fonction `fflush` peut être utilisée pour vider un fichier puisque le seul paramètre

qu'elle prend en compte correspond à une variable de type `FILE`.

Lecture d'un fichier

Les fonctions qui prennent en paramètre un fichier correspondant à un flot de données sont tout à fait adaptées à la lecture du contenu d'un fichier classique autre que celui du flot d'entrée standard (`stdin`).

Le code simplifié ci-dessous présente un exemple de lecture d'un fichier contenant du texte et dont les lignes n'excèdent pas quatre-vingt caractères :

```
char lig[81];
FILE *fic;

while (fgets(lig, 81, fic) != NULL)
  printf("%s", lig);
```

Le fichier pointé par la variable `fic` doit être préalablement ouvert en lecture pour que ce code puisse s'exécuter correctement. Les lignes sont successivement stockées jusqu'à la fin de fichier dans la variable `lig` pour être enseuite affichées.