CRO : Addentum sur le `scanf` et les caractères

1 Introduction

L'utilisation de la fonction `scanf` est assez complexe, le TD2 tentait d'en proposer une version simple mais n'était pas clair. Ce document tente de remettre de l'ordre dans les idées sur le `scanf` et la lecture au clavier en général (et en particulier la lecture de caractères). J'ai utilisé des discussions avec les gens du Citi et la page `https://xrenault.developpez.com/tutoriels/c/scanf/` pour ce document.

2 scanf

On a compris que a fonction `int scanf(const char *format, ...)` prend en premier argument une chaîne de caractère qui contient les formats désignant le type des objets que l'on va lire au clavier. Par exemple :`scanf("%d",&i)`

On a aussi compris que tout ce que l'on tape au clavier est stocké dans un buffer d'entrée (`stdin`) dès que l'on tape un retour chariot. La lecture d'un nombre avec `scanf` **laisse le retour chariot dans le buffer d'entrée**, donc on a le comportement suivant en supposant que l'utilisateur entre 9 puis retour au clavier :

ce programme :

```
printf("entrez un entier: ");
scanf( "%d", &i);
printf("entier lu: %d\n",i);
printf("entrez un caractere");
scanf("%c",&c);
printf("caractere lu: %c\n",c);
```

donne le résultat :

```
entrez un entier   9
entier lu: 9
entrez un caracterecaractere lu:
```

Le comportement de `scanf` diffère selon que la donnée lue est un caractère, une chaîne de caractère ou d'un autre type. si la donnée n'est pas un caractère (par exemple un entier), les espaces (ou les retour chariot d'ailleurs) précédent l'entier ou entre les entiers lus vont être ignorés. Par exemple :

ce programme :

```
printf("entrez deux entiers: ");
scanf( "%d%d", &i, &j);
printf("entiers lu: i=%d, j=%d\n",i,j);
```

donne le résultat :

```
entrez deux entiers: 8   9
entiers lu: i=8, j=9
```

En revanche, si on cherche à lire des caractères, l'espace et le retour chariot étant des caractères, ils sont lus comme tels, d'où le comportement du premier exemple ci-dessus : le caractère lu est le retour chariot qui suivait l'entier dans le buffer d'entrée, c'est bien lui qui est affiché.
Dernier comportement un peu spécial : la lecture de chaînes de caractère. Les "chaînes de caractères" en C sont des tableaux de caractères ou le caractère terminant la chaîne est '\0', c'est à dire un octet de valeur 0. Lorsque l'on utilise le format `%s` avec `scanf`, le programme lit la chaîne de caractères et *rajoute un caractère '\0' à la fin*, toujours en laissant le retour chariot dans le buffer d'entrée.

ce programme :

```
char chaine[10];

printf("entrez une chaîne: ");
scanf( "%s", chaine);
printf("Chaîne lue: %s\n",chaine);
printf("entrez un caractere");
scanf("%c",&c);
printf("caractere lu: %c\n",c);
```

donne le résultat :

```
entrez une chaîne: yuiyui
Chaîne lue: yuiyui
entrez un caracterecaractere lu:
```

3 Comment lire des caractères

Il y une solution simple à ce problème : mettre un espace dans la chaîne de format avant le %c, et deux méthodes toutes deux un peu lourdes : soit on utilise des expressions régulières dans la chaîne de contrôle de `scanf` pour "ignorer" les caractères que l'on veut ignorer, soit on vide explicitement le buffer d'entrée avec la fonction `getc`

3.1 Méthode simple

Si on met un espace avant le format `%c` dans la chaîne de format, cet espace "avale" tous les espaces avant le caractère mais aussi tous les caractères "blancs" (tabulation ou retour chariot). C'est la méthode la plus simple mais il faut connaître le truc. Exrait de `https://cplusplus.com/reference/cstdio/scanf/` :

```
int scanf ( const char * format, ... );
format: C string that contains a sequence of characters that control
how characters extracted from the stream are treated:

- Whitespace character: the function will read and ignore any
whitespace characters encountered before the next non-whitespace
character (whitespace characters include spaces, newline and tab
characters -- see isspace). A single whitespace in the format
string validates any quantity of whitespace characters extracted
from the stream (including none).
[...]
```

3.2 Vider le buffer d'entrée

La méthode standard pour vider le buffer d'entrée jusqu'au retour chariot est d'utiliser le code suivant (`EOF` signifie *End of file* et indique donc qu'il n'y a plus rien à lire dans le buffer d'entrée) :

```
int c;
while ( ((c = getchar()) != '\n') && c != EOF)
{
};
```

ce programme :

```
printf("entrez un entier: ");
scanf( "%d", &i);
printf("entier lu: %d\n",i);
while ( ((c = getchar()) != '\n') && c != EOF)
  {
  };
printf("entrez un caractere");
scanf("%c",&c);
printf("caractere lu: %c\n",c);
```

donne le résultat :

```
entrez un entier: 8
entier lu: 8
entrez un caractereh
caractere lu: h
```

Notez que la boucle lit aussi le retour chariot, pas besoin donc de rajouter un `getchar()` après la boucle.

3.3 Utiliser les expressions régulières dans `scanf`

Cela nécessite évidement de savoir ce qu'est une expression régulière, ce que je ne vais pas expliquer ici. `scanf` propose une syntaxe pour des expressions régulières qui vont lui permettre d'ignorer les caractères "reconnus" par l'expression régulière. par exemple le format : `%*[^\n]`

signifie : je reconnaît n'importe quel caractère sauf un retour chariot (avec "^") et j'ignore cette séquence (avec le *). on a donc le comportement suivant :

ce programme :

```
printf("entrez un entier: ");
scanf( "%d%*[^\n]", &i);
printf("entier lu: %d\n",i);
getchar();// retour chariot
printf("entrez un caractere");
scanf("%c",&c);
printf("caractere lu: %c\n",c);
```

donne le résultat :

```
entrez un entier: 9 qsdqkjdl
entier lu: 9
entrez un caracterec
caractere lu: c
```

Notez bien que j'ai tapé plein de caractères après l'entier (et avant le retour chariot), ces caratères sont ignorés. Le retour chariot lui, reste dans le buffer, il faut dont le "manger" avec un `getchar()`. L'avantage de ces deux méthodes (par rapport à simplement mettre un `getchar()` dans le premier programme pour manger le retour chariot), c'est que l'on est sûr de tout manger, notamment les espaces si il y en a après l'entier.

4 Pour les chaîne de caractères : `fgets`

Dernière chose, il est maintenant fortement recommandé de ne pas utiliser la fonction `gets` pour lire une chaîne de caractère. En effet, cette fonction est à l'origine des *buffer overflow* qui permettent très facilement de faire des attaques malicieuses. `gets` ne demande pas explicitement la taille de la chaîne lue au clavier. En entrant une chaîne plus grande que le tableau dans lequel elle doit être stockée, on peut écraser des variables du programme ou de l'environnement d'exécution et provoquer un comportement anormal.
La fonction `char *fgets(char *s, int size, FILE *stream)` elle, demande explicitement la taille maximale de la chaîne lue (ne pas oublier que cette taille inclut le '\0' qui est mis en place par `fgets` à la fin de la chaîne).

ce programme :

```
char chaine[7];

printf("entrez une chaîne: ");
fgets(chaine,6,stdin);
printf("chaine lue: %s\n",chaine);
```

donne le résultat :

```
entrez une chaîne: bonjour
chaine lue: bonjo
```