INFORMATIQUE
Sciences
Université Paris Cité

Langage C –Cours 9

Lélia Blin & Pierre Charbit

prenom.nom@irif.fr

2023 - 2024

Saisie de données sécurisée

Retour sur certaines fonctions

Bonnes pratiques

- C un langage de base
- Une mauvaise déclaration de données (au clavier ou en fichier) peut entrainer
  - Une mauvais fonctionnement du code attendu
  - Un débordement mémoire → utilisé pour faire des attaques

Entrée standard

- Données entrantes (`stdin` standard input)
  - Clavier
  - fichier
  - ...

Fonctions `gets()`

Ancienne fonction qui permet de lire une chaîne de caractères depuis le clavier

```c
#include <stdio.h>
#include <string.h>

int main(void) {
        char nom[8]; // Déclaration d'un tableau de caractères pour stocker le nom
        printf("Entrez votre nom : ");
        gets(nom); // Lecture du nom à partir de l'entrée standard
        printf("Bonjour, %s !\n", nom); // Affichage du nom saisi

        return 0;
}
```

> aucune sécurité (on peut écrire plus que la taille prévue) -> A ne pas utiliser

Bonnes pratiques

- prévoir tous les cas que peut générer un utilisateur
  - Bienveillant mais peut adroit
  - malveillant

Retour sur `scanf`

pas sécurisé mais... on peut limiter les dégâts

Exemple

```c
//Scanf_age_1.c
#include <stdio.h>

int main(void) {
        int age = 0;
        printf("Quel âge avez-vous? ");
        scanf("%d", &age);
        printf("Vous avez dit: %d ans\n", age);
        return 0;
}
```

Regardons les retours

```
./Scanf_age_1
Quel âge avez-vous? 5
Vous avez dit: 5 ans
./Scanf_age_1
Quel âge avez-vous? lkjh
Vous avez dit: 0 ans
./Scanf_age_1
Quel âge avez-vous? 444444444444444444444444
Vous avez dit: -1 ans
```

Récupération des retours pour tester

```c
#include <stdio.h>

int main(void) {
        int age = 0;
        int ret;

        printf("Quel âge avez-vous? ");
        ret = scanf("%d", &age);

        printf("Vous avez dit: %d ans\n", age);
        printf("ret : %d\n", ret);

        return 0;

}
```

```
Quel âge avez-vous? 5
Vous avez dit: 5 ans
ret : 1
./Scanf_age_2
Quel âge avez-vous? lhgjhgl
Vous avez dit: 0 ans
ret : 0
./Scanf_age_2
Quel âge avez-vous? 444444444444444444444
Vous avez dit: -1 ans
ret : 1
```

Aller plus loin avec les retours

- On peut tester en entrant plusieurs données

```c
//Scanf_age_poids.c
#include <stdio.h>

int main(void) {
    int age = 0, taille = 0, poids = 0;
    int ret;

    printf("Indiquer votre age - votre taille en cm - votre poids (en kilo) ");
    ret = scanf("%d - %d - %d", &age,&taille,&poids);

    printf("Info perso: %d ans - %d cm - %d kg\n", age,taille,poids);
    printf("ret : %d\n", ret);

}
```

Retours

```
./Scanf_age_poids
Indiquer votre age - votre taille en cm - votre poids (en kilo) 12 - 155 - 45
Info perso: 12 ans - 155 cm - 45 kg
ret : 3
```

- Bonnes données au bon formatage
  - 3 sur 3 (ret=3)

Retours

```
./Scanf_age_poids
Indiquer votre age – votre taille en cm – votre poids (en kilo) A – 10 – 12
Info perso: 0 ans – 0 cm – 0 kg
ret : 0
```

- Des que scanf rencontre une mauvaise donné il s'arête
  - 0 sur 3

Retours

```
./Scanf_age_poids
Indiquer votre age – votre taille en cm – votre poids (en kilo) 12 /12 /12
Info perso: 12 ans – 0 cm – 0 kg
ret : 1
```

- pas le bon formatage après une bonne donnée

Retours

```
./Scanf_age_poids
Indiquer votre age - votre taille en cm - votre poids (en kilo) 12 - A -12
Info perso: 12 ans - 0 cm - 0 kg
ret : 1
./Scanf_age_poids
Indiquer votre age - votre taille en cm - votre poids (en kilo) 12 - 12 - A
Info perso: 12 ans - 12 cm - 0 kg
ret : 2
```

Mais le retour ne vérifie pas la taille de l'information saisie

Chaines de caractères

```c
//Scan_string_1.c
#include <stdio.h>

#define string_max 5
int main(void) {

    char chaine[string_max+1]; //ne pas oublier \0
    printf("mot en 5 lettres : ");
    scanf("%s",chaine);

    return 0;
}
```

Utilisations

```
./Scanf_string_1
mot en 5 lettres : blin

./Scanf_string_1
mot en 5 lettres : antituconstitutionnellement
zsh: bus error  ./Scanf_string_1
```

Contrôler la taille des données

- rajouter du formatage

```c
//Scanf_string_2.c
#include <stdio.h>

#define string_max 5
int main(void) {

    char chaine[string_max+1]; //ne pas oublier \0
    printf("mot en 5 lettres : ");
    scanf("%5s",chaine);
    printf("Mot saisi: %s\n",chaine);

    return 0;
}
```

Retours

```
./Scanf_string_2
mot en 5 lettres : antituconstitutionnellement
Mot saisi: antit
```

- la fonction lit 5 caractères
- le reste est ignoré

oui mais...

```
./Scanf_string_2
mot en 5 lettres : le ka
Mot saisi: le
```

- ne pas oublier les caractères spéciaux
- `scanf` s'arête quand il rencontre un espace

Attention aux scanf successifs 1

```c
int main(void) {
        int x=0,y=0;
        printf("Un premier entier?\n");
        scanf("%d",&x);
        printf("Un second entier?\n");
        scanf("%d",&y);
        printf("--x=%d/y=%d--",x,y);
    return 0;
}
```

- Si on tape `12` puis `34` pas de souci, on voit `--x:12/y:34--`
- Si on tape `12aa`, on n'a meme pas la main pour taper dans le deuxième `scanf` et on voit `--x=12/y=0--` ??!??

Attention aux scanf successifs 1 : Explication

- Quand `scanf` rencontre dans stdin quelquechose qui ne correspond pas au caractère demandé, il s'arrête mais les caractères restent dans le tampon, prêts à être consommés par le `scanf` suivant.
- Ceci vaut aussi pour le `\n` tapé pour finir la saisie du premier `scanf`
- Ici le `aa\n` est resté, le deuxieme `scanf` arrive et retombe dessus, ressort car le format n'est pas bon (il attend un `int`)
- `y` n'a pas été modifié (et le buffer contient toujours `aa\n`)

Attention aux scanf successifs 2 :

```c
int main(void) {
        char c1,c2;
        printf("Un premier caractère?\n");
        scanf("%c",&c1);
        printf("Un second caractère?\n");
        scanf("%c",&c2);
        printf("--c1=%c/c2=%c--\n",c1,c2);
    return 0;
}
```

Si on tape `a` puis Entrée on n'a pas la main pour le second `scanf` et on voit

```
--c1=a/c2=
--
```

Attention aux scanf successifs 2 : Explication

Explication : le `\n` est resté dans le buffer et il est mangé par le deuxieme `scanf` (c'est bien un char) qui le stocke dans c2

Attention aux scanf successifs 1 : retour

```c
int main(void) {
        int x=0,y=0;
        printf("Un premier entier?\n");
        scanf("%d",&x);                          // on tape 12 puis entrée
        printf("Un second entier?\n");
        scanf("%d",&y);                          // on tape a
        printf("--x=%d/y=%d--",x,y);             // on voit --x=12/c=a--
        return 0;
}
```

Mais alors, si le `\n` est resté dans le buffer après le premier `scanf`, pourquoi le premier exemple ci dessus a bien fonctionné?

Il aurait du être consommé par le second `scanf` non?

Attention aux scanf successifs 1 : explications

Explication : `%d` (comme `%f` pour les flottants) a un fonctionnement particulier, il **ignore tous les caractères d'espacement en début de saisie** (comme `' '` ou `\n`, mais aussi `'\r'`, `'\t'` et `'\v'`).

Ce n'est pas le cas de `%c` qui traite les espaces ou les `\n` comme tous les autres caractères.

Attention aux scanf successifs 2 : retour

Mais alors comment faire pour résoudre le problème ici et saisir deux caractères??

```
int main(void) {
        char c1,c2;
        printf("Un premier caractère?\n");
        scanf("%c",&c1);
        printf("Un second caractère?\n");
        scanf("%c",&c2);
        printf("--c1=%c/c2=%c--\n",c1,c2);
        return 0;
}
```

Attention aux scanf successifs 2 : retour

```
int main(void) {
        char c1,c2;
        printf("Un premier caractère?\n");
        scanf("%c",&c1);
        printf("Un second caractère?\n");
        scanf(" %c",&c2);                          //Notez l'espace !!
        printf("--c1=%c/c2=%c--\n",c1,c2);
        return 0;
}
```

On met un **espace** avant le %c dans le second `scanf` !!

Cela fait alors comme pour `%d` : tous les caractères d'espacement (donc le `\n` resté sur le buffer) sont ignorés.

Formater un peu plus ...

.... Expressions régulières

"Expressions régulières"

- On utilise l'indicateur de formatage avec crochets `%[...]`
- Ce ne sont pas des vraies expressions regulières regexp, elles sont moins expressives
- Si on écrit `%[aei]` cela signifie "une séquence de caractères ne comprenant que `a` `e` ou `i`"
- Le scan s'arrête dès qu'on recontre un caractère qui n'est pas dans la liste.
- On peut ecrire `%[^aei]` pour signifier "une séquence de caractères tous distincts de `a` `e` ou `i`
- On peut utiliser un tiret pour spécifier un intervalle de caractères.
  Par exemple `%[0-9a-z]` pour siginier une séaquence de caractères composée de caractères numériques ou de lettre minuscules

```c
//Scanf_string_3
#include <stdio.h>

#define string_max 5
int main(void) {

    char chaine[string_max+1]; //ne pas oublier \0
    int ret;

    printf("aimez vous la littérature?  ");
    ret = scanf("%[oui]",chaine);

    printf("réponse saisi: %s\n",chaine);
    printf("ret: %d\n",ret);

    return 0;
}
```

Retour

```
aimez vous la littérature?  non
réponse saisi:
ret: 0
./Scanf_string_3
aimez vous la littérature?  je pense que oui
réponse saisi:
ret: 0
./Scanf_string_3
aimez vous la littérature?  oui
réponse saisi: oui
ret: 1
 ./Scanf_string_3
aimez vous la littérature?  uio
réponse saisi: uio
ret: 1
```

Fonction sécurisé ... `fgets()`

La fonction `fgets()`

- fonction de la bibliothèque standard du langage C, déclarée dans l'en-tête `stdio.h`.

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

- `str` : est un pointeur vers le tampon (buffer) où stocker la chaîne lue.
- `n` : spécifie le nombre maximal de caractères à lire, y compris le caractère nul de fin de chaîne. Cela évite le dépassement de tampon (buffer overflow).
- `stream` : est un pointeur vers le flux de données à partir duquel lire la chaîne. Il peut s'agir d'un fichier ou de `stdin` pour l'entrée standard.

La fonction `fgets()`

- La fonction `fgets()` lit une ligne entière (y compris le retour à la ligne `\n`) depuis le flux spécifié jusqu'à ce qu'elle rencontre un caractère de nouvelle ligne, un EOF (fin de fichier) ou que `n - 1` caractères aient été lus.
- Elle stocke ensuite la chaîne lue dans le tampon `str`, en ajoutant un caractère nul de fin de chaîne à la fin.
- Il est important de noter que `fgets()` conserve le caractère de nouvelle ligne `\n` s'il est présent dans le fichier ou dans l'entrée standard.

Retour

- La fonction renvoie `NULL` si une erreur se produit ou si la fin de fichier est atteinte avant que `n - 1` caractères ne soient lus.
- Sinon, elle renvoie un pointeur vers la chaîne lue.

Exemple

```c
#include <stdio.h>

#define string_max 5

int main(void) {

    char chaine[string_max+1]; //ne pas oublier \0
    printf("mot en 5 lettres : ");
    fgets(chaine,string_max+1,stdin);
    printf("Mot saisi: %s\n",chaine);

    return 0;
}
```

Retour

```
./fgets_1
mot en 5 lettres : blin
Mot saisi: blin
./fgets_1
mot en 5 lettres : charbit
Mot saisi: charb
./fgets_1
mot en 5 lettres : T tot
Mot saisi: T tot
```

- `fgets` ne s'arrête pas aux espaces.

```c
//fgets_2
#include <stdio.h>

#define string_max 5

int main(void) {

    char chaine[string_max+1]; //ne pas oublier \0

    printf("mot en 5 lettres : ");
    fgets(chaine,string_max+1,stdin);
    printf("Mot saisi: %s\n",chaine);

    printf("--------------\n");

    printf("mot en 5 lettres : ");
    fgets(chaine,string_max+1,stdin);
    printf("Mot saisi: %s\n",chaine);

    return 0;
}
```

```
 ./fgets_2
mot en 5 lettres : tests
Mot saisi: tests
--------------
mot en 5 lettres :
Mot saisi:

./fgets_2
mot en 5 lettres : testsblin
Mot saisi: tests
--------------
mot en 5 lettres : Mot saisi: blin
```

La fonction reprend l'information qui reste dans le tampon de lecture.

Le tampon d'entrée (`stdin`)

- tampon d'entrée `stdin` est une mémoire temporaire utilisée par le système d'exploitation pour stocker les données entrantes provenant des périphériques d'entrée standard, comme le clavier.
- Lorsque vous saisissez à l'aide de fonctions telles que `scanf()` ou `fgets()`,
  - ces données sont d'abord stockées dans le tampon `stdin`.
  - Ensuite, votre programme peut lire ces données depuis ce tampon.

Qui gère ce tampon?

- le tampon `stdin` est généralement géré par le système d'exploitation et n'est pas directement accessible ou modifiable par le programme C lui-même.

> Comment faire pour vider le tampon?

Vider le tampon

- pour cela il faut : Lire et ignorer les caractères du tampon jusqu'à ce qu'un `\n` soit rencontré

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

- voir le code `fgets_4.c`.

Conclusion

- Sécuriser votre code quand vous récupérez des données extérieures