L3 MI — Programmation

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://membres-ljk.imag.fr/Pierre.Karpman/tea.html

2022–10–19

Chaînes de caractères

Arguments du `main`

Chaînes de caractères en C : type `char`

Le type `char` correspond à un entier de 8 bits (dont le caractère signé ou non dépend de l'implémentation)

- On peut par exemple faire `char x = 12;`
- Mais il possède son propre format d'affichage : `printf("%c",c)` $\rightsquigarrow$ affichage du caractère ASCII dont le code correspond à la valeur de c
- Il existe aussi des littéraux pour les caractères affichables, par ex.

```
char c1 = 'a';
char c2 = '\n'; // retour à la ligne
char c3 = '\t'; // tabulation
char c4 = '\0'; // <=> char c4 = 0;
```

- On peut « calculer » avec, par ex. `'<' + '<'; // == 'x'`

Chaînes de caractères en C : type `char *`

Une chaîne de caractère est de type `char *`

- C'est un simple pointeur vers des `char`
- Mais avec une sémantique particulière par rapport à beaucoup de fonctions
- Il possède son propre format d'affichage : `"%s"`
- Et ses propres littéraux : `"coucou"`
- ¿¿¿¿ ATTENTION : le bon fonctionnement de ces cas particuliers requiert généralement que la chaîne se termine par `'\0'` ????
- Ce caractère est automatiquement ajouté dans un littéral `"comme celui ci"`

Null-terminated string

- Rappel : en C, la taille d'une zone mémoire à une dimension accessible via un pointeur (par ex. `int *t`)
  - Ne fait pas partie du type `int *`
  - N'est pas accessible depuis la variable `t`
- Pour une chaîne de caractère `char *s`, la longueur de chaîne est implicitement comprise *par les fonctions usuelles de manipulation de chaînes* comme min(`i`) t.q. `s[i] == '\0'`

Exemples

```
char s[13] = {'c', 'o', 'u', 'c', 'o', 'u', '\0',
↪  'p', 'o', 'n', 'e', 'y', '\0'};
```

- `printf("%s",s)` ⇝ coucou
- `printf("%s",s+1)` ⇝ oucou
- `printf("%s",s+6)` ⇝
- `printf("%s",s+7)` ⇝ poney

On aurait aussi pu déclarer `s` comme :

```
char s[13] = "coucou\0poney";
```

- Le dernier `'\0'` est automatiquement ajouté
- Mais il prend bien un octet ! ⇝ `s` a une taille de 13 !

Les chaînes de caractère en C sont dangereuses !

- Allocation délicate (penser à la place pour `'\0'`)
- Bugs « faciles » aux conséquences graves
  - Dépassement de tampon $\rightsquigarrow$ possibles failles de sécurité
- Idéalement : ne jamais utiliser C pour manipuler de tels objets

`string.h`

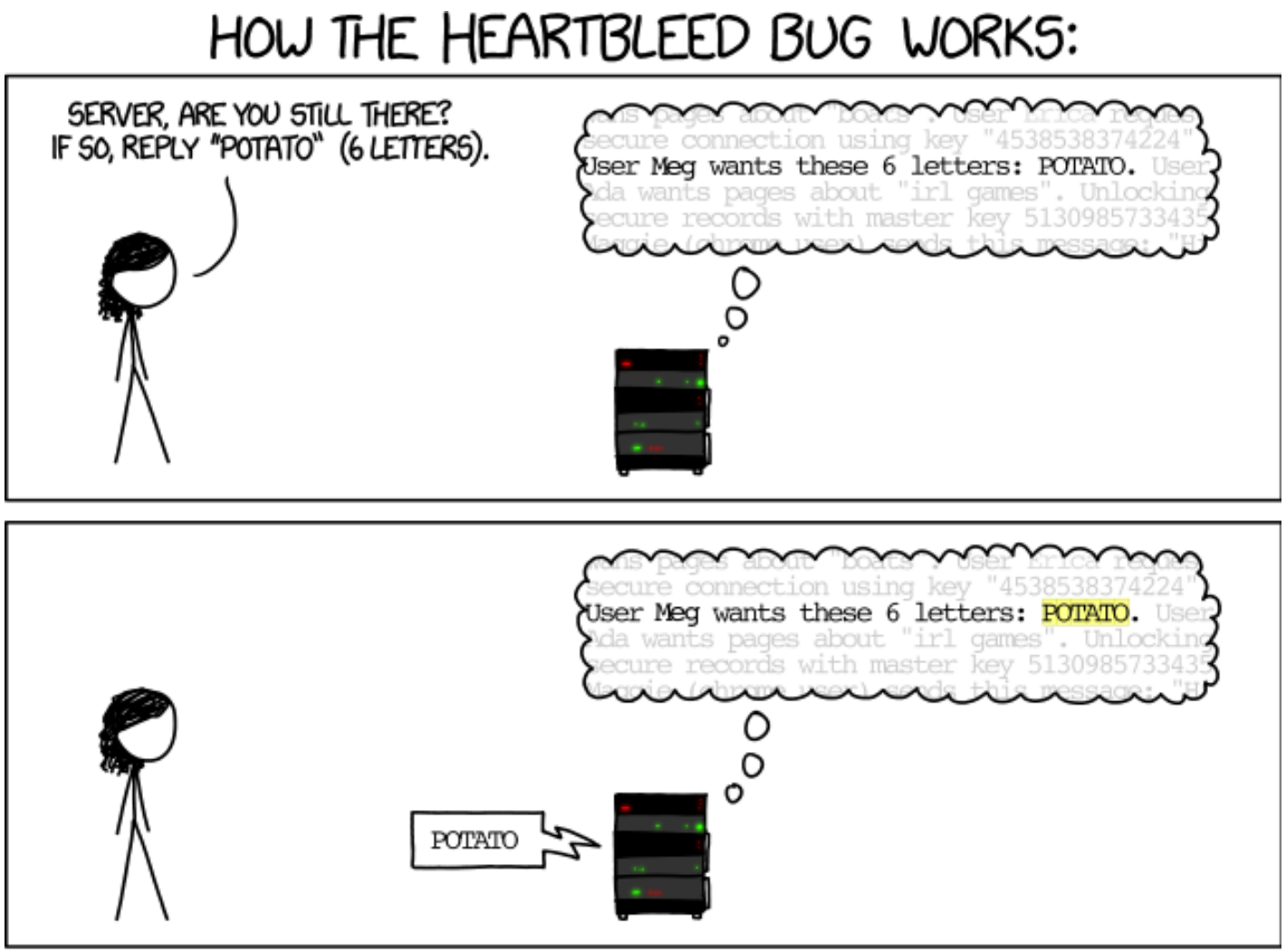
Il existe une bibliothèque standard de traitement de chaînes en C, déclarées dans `string.h`

- Documentées *sur internet* dans les pages man
- ¿¿¿¿ *The string functions manipulate strings that are terminated by a null byte* ????

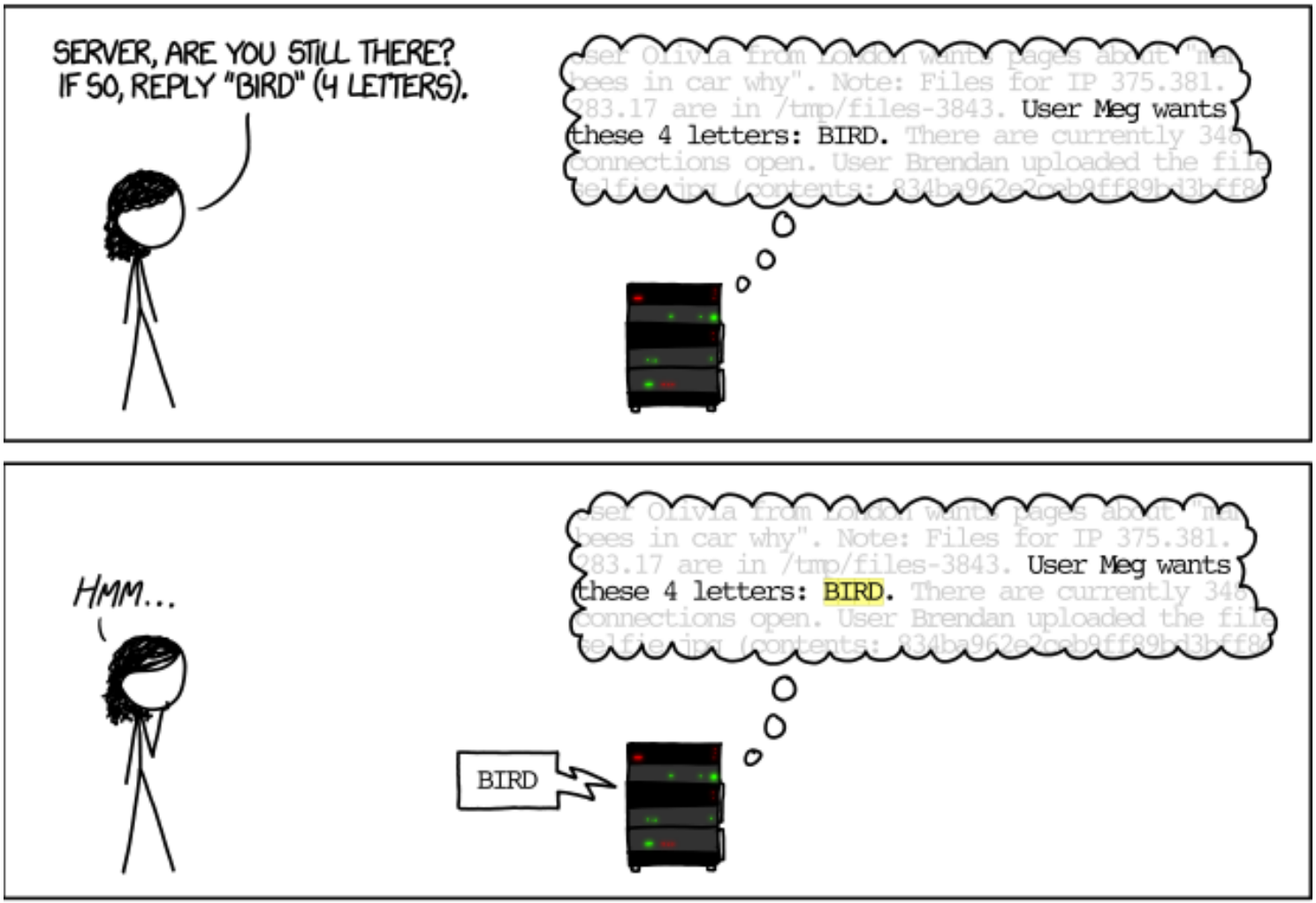
Quelques exemples

```
size_t strlen(const char *s); // 'Finding the length
↪  of a string in Pascal is one assembly instruction
↪  instead of a whole loop.' ; dangereux
char * strcpy(char * dst, const char * src); // Fait
↪  une copie ; dangereux !
char * strncpy(char * dst, const char * src, size_t
↪  len); // Fait une copie ; aussi dangereux (mais
↪  moins) !
int strncmp(const char *s1, const char *s2, size_t
↪  n); // Comparaison
```

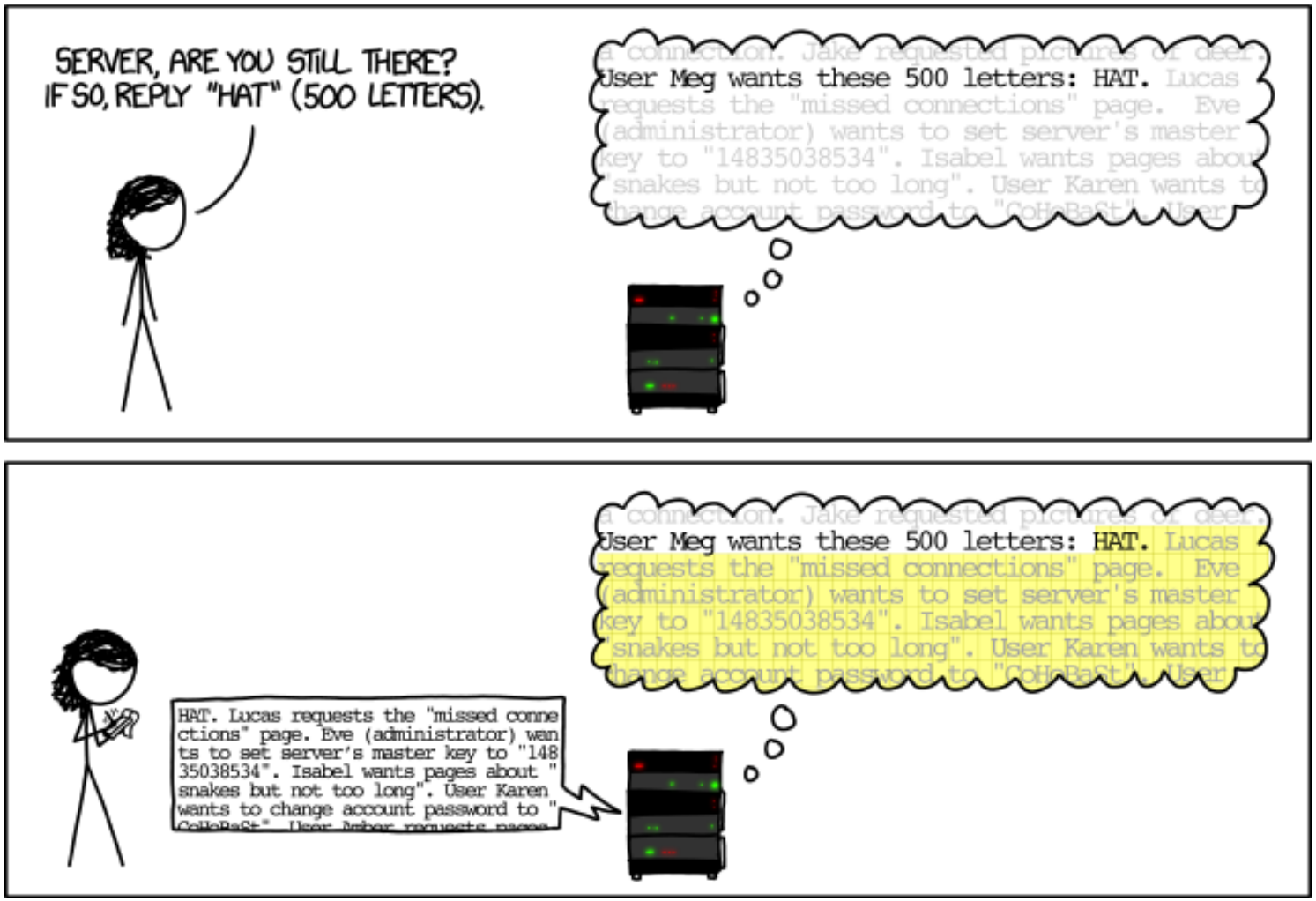
Un problème délicat... (1) — https://xkcd.com/1354/



Un problème délicat... (2) — https://xkcd.com/1354/



Un problème délicat... (3) — https://xkcd.com/1354/



Chaînes de caractères

Arguments du `main`

Rappel de prototype

De façon générale, le `main` a le prototype
`int main(int argc, char **argv)` (ou `char *argv[]`)

- `argc` donne le nombre d'arguments passés au programme en ligne de commande
- `argv` est un tableau de `argc` chaînes de caractères qui contient chaque argument
- `argv[0]` est le nom avec lequel le programme est invoqué

Exemples

petit.c :

```
int main(int argc, char **argv)
{
        printf("%s %s\n", argv[0], argc > 1 ? argv[1]
        ↪  : "");
        return 0;
}
```

```
> cc -o petit petit.c
> ./petit poney
./petit poney
> /usr/pierre/sw/petit
/usr/pierre/sw/petit
```

Limites...

- La taille et le nombre max. d'arguments pouvant être passés est limitée par le système
- Sous UNIX, cette valeur peut se trouver (comme d'autres du même genre) dans `limits.h`
- Ou via `getconf` en ligne de commande

Arguments++

On peut utiliser le prototype alternatif

```
int main(int argc, char **argv, char **envp)
```

- `envp` est un tableau de chaînes de caractères terminé par `NULL` qui contient les variables d'environnement
- Variante portable : la fonction `getenv`

Exemple d'accès :

```
int main(int argc, char **argv, char **envp)
{
        for (int i = 0; printf("%s\n", envp[i] ==
        ↪  NULL ? exit(0),"" : envp[i]) ; i++);
        return 0;
}
```

La fonction `exit`, déclarée dans `stdlib.h`, quitte le programme avec le code de retour passé en argument