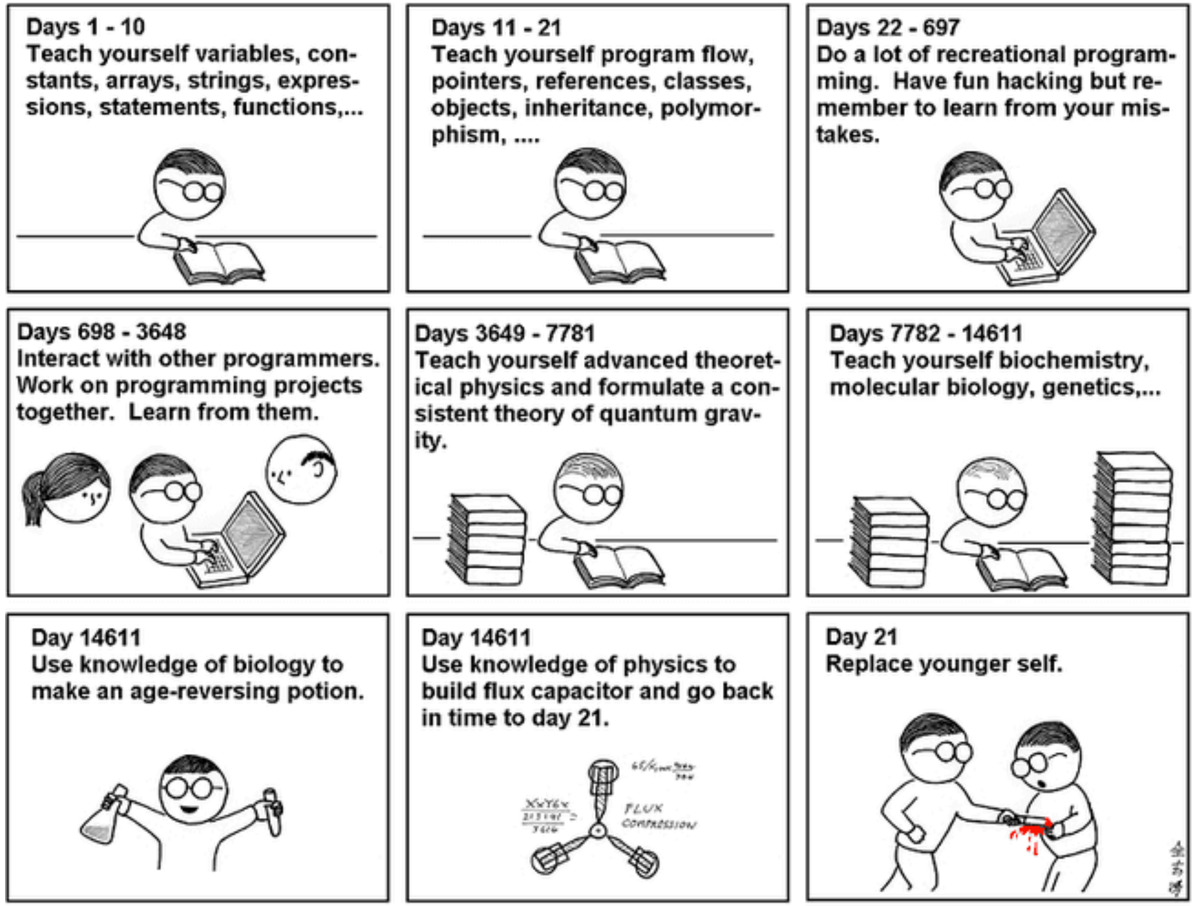
Introduction à la programmation

ICC-C: Cours 1



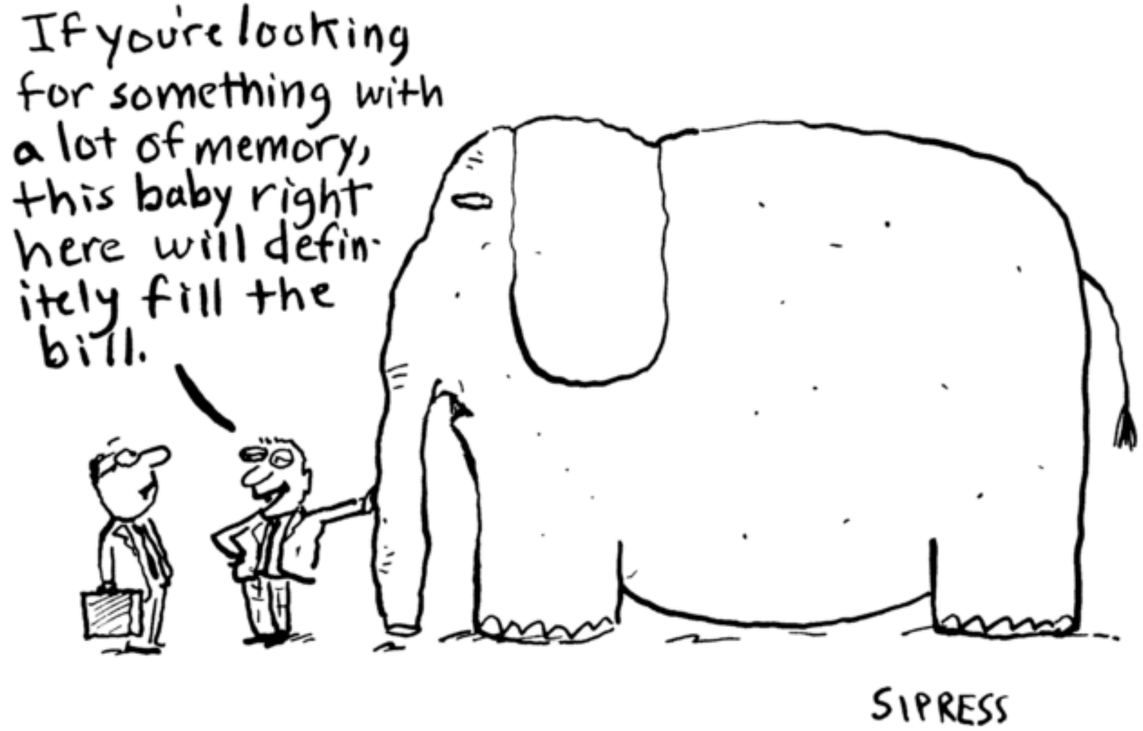
https://abstrusegoose.com/249

Dan Tomozei

Qu’est-ce qu’un ordinateur?

Ce n’est pas une question piège…

- Unité centrale (*CPU*)
- Disque / stockage (*SSD, HDD*)
- Mémoire vive (*RAM*)



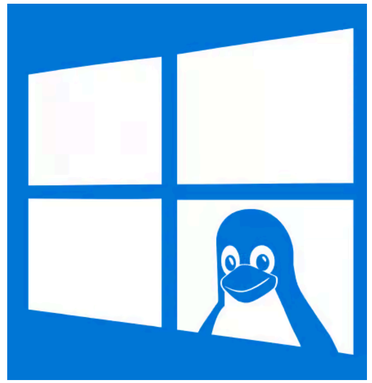
David Sipress | CartoonStock.com

Qu’est-ce qu’un programme?

- Un programme est une suite d’instructions qui sont exécutées par l’ordinateur
- Souvent un programme est encodé dans un fichier binaire exécutable stocké sur disque
- Quand on lance un programme
  - l’exécutable est chargé en mémoire (*RAM*)
  - les instructions sont exécutées par l’unité centrale (*CPU*)
  - souvent on traite des données en mémoire avant de les stocker sur disque

L’environment Unix

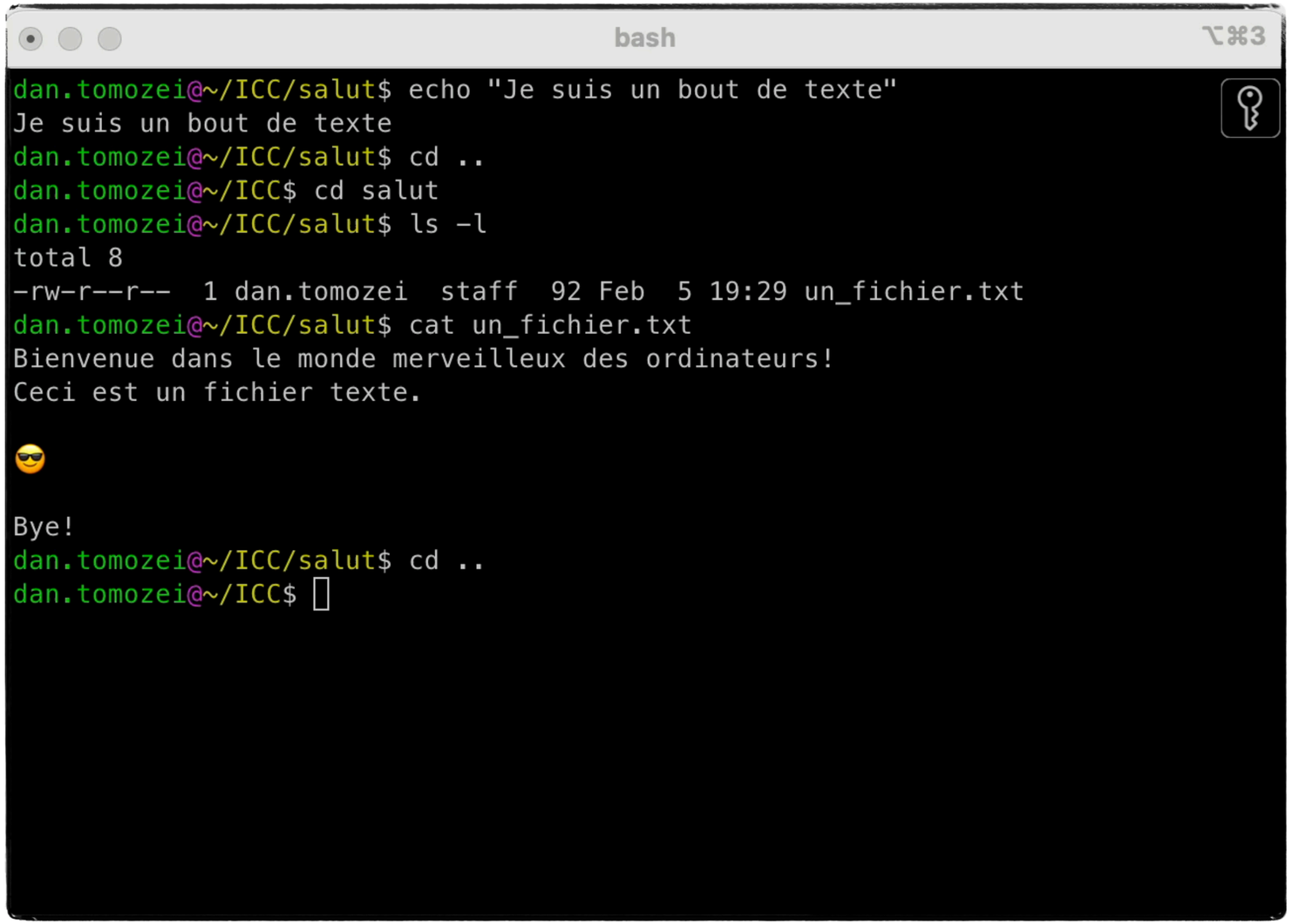
- Dans ce cours nous utilisons des systèmes Unix
  - (Ubuntu) Linux
  - MacOS
  - Windows / WSL



Le terminal

Command Line Interface

- Le terminal est un outil puissant pour interagir avec l'ordinateur
- Il comprend des instructions
- E.g., avec `cd` (="change directory") on peut naviguer dans le système de fichiers
- Il permet de lancer d'autres programmes
- E.g., le programme `cat` qui affiche un fichier à l'écran



Comment écrit-on un programme C?

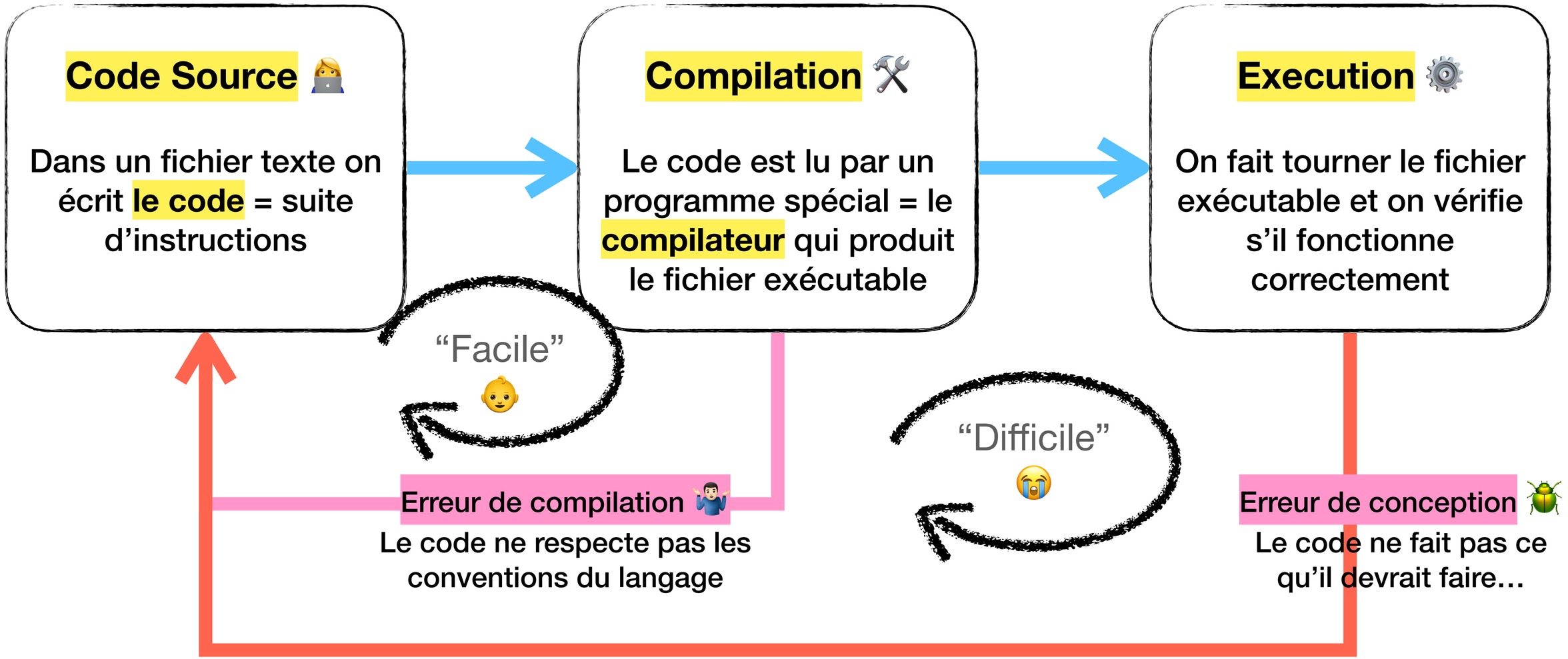
Le code source

- Fichiers texte avec extension (=qui finissent en) `.c`
- Le code source est composé d'instructions regroupées dans des fonctions
- La fonction `main()` est spéciale dite "d'entrée" (*entry point*)
- Elle set exécutée quand on lance le programme

```c
#include <stdio.h>

int main()
{

    votre code ici

}
```

Comment écrit-on un programme?

“The dev cycle”



Hello, World!

- Votre premier programme!
- Contient une seule instruction
- Affiche à l'écran

```
Hello, World!
```

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
}
```

- Après la fonction `main`,
  voici la fonction `printf` qui fait afficher du texte!

Compilation du code source

- Nous utiliserons le compilateur `gcc`
- GNU Compiler Collection
- … avant c'était "GNU C Compiler" 👴🏻
- Pour lancer le compilateur:

```
gcc <fichier_source> -o <nom_executable>
```

Votre code

Sera créé si tout se passe bien

Que fait le compilateur C?

- Vérifie que le code source satisfait la syntaxe du langage C
- Si non, il produit une erreur de compilation ❌
  - 🤖 “A la ligne *x* de votre code je ne comprends pas ce que veut dire *y*”
- Si oui, alors il traduit les instructions en “langage machine”
- … et produit le fichier executable
  - Spécifique à votre CPU + système d’exploitation
  - On ne peut pas faire tourner un executable Mac sur Linux, etc.

Compilation et exécution

hello.c

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
}
```

- Dans le terminal
  `gcc hello.c -o hello`
- On passe comme paramètres
  - le nom du fichier source `hello.c`
  - `-o` suivi du nom du fichier exécutable à produire `hello`
- Si tout se passe bien, il n'y a rien qui s'affiche
- On lance notre nouveau programme depuis le terminal avec
  `./hello`
- Au terminal on voit s'afficher le texte
  `Hello, World`

Commentaires

Le compilateur ne les lit pas

```c
#include <stdio.h>

/*
Je suis un commentaire multi-ligne.
Je peux être aussi long que je veux.
Je peux même contenir des caractères spéciaux comme " et \.
Et n'oublions pas les emojis: 🐵
*/

int main()
{
    printf("Hello, World!\n"); // Je suis un commentaire sur une seule ligne
    printf("Bonjour, ICC!\n"); // Je suis un autre commentaire sur une ligne
}
```

Le langage C

Variables et constantes

DCT, 2023.12

Valeurs constantes

- En C on peut écrire des valeurs constantes, par exemple:
  - des *entiers* en base 10:
    ```
    1, 2, 3;
    ```
  - des *entiers* en d'autres bases (8, 16):
    ```
    021103, 0xa, 0xce1;
    ```
  - des nombres *réels*:
    ```
    -13.2, 5.75, 1e-5;
    ```
  - des *chaînes de caractères*:
    ```
    "J'aime les ordinateurs";
    ```

Opérateurs et expressions

- On peut leur appliquer toutes sortes d'opérateurs
  - unaires:
    ```
    -1, ~0x7;
    ```
  - binaires:
    ```
    5 + 6, 11 / 3, 0xff & 0xa, 1 || 0;
    ```
  - ternaires:
    ```
    1 ? 2 : 3;
    ```
- On obtient des expressions simples

Opérateurs et expressions

- … mais attention aux opérandes !

```
    3 * "abc";
```

```
values.c:17:7: error: invalid operands to binary expression ('int' and 'char[4]')
    3 * "abc";
    ~ ^ ~~~~~
1 error generated.
```

- Pourquoi?
  - en C on ne peut pas multiplier un entier et une chaîne — les types sont incompatibles!

Qu’est-ce qu’un type?

- Un type est une propriété des valeurs qu’on manipule dans un langage
- Chaque valeur a un type (entier, réel, etc.) déterminé par le compilateur
- Ainsi, le compilateur peut valider certaines parties du code source
- Les types aident à éviter des ***erreurs de conception***

Types numériques

- `int` = Le type par défaut pour les valeurs *entières*
  - Si on écrit `100`, le compilateur “crée” une valeur constante de type `int`
- `double` = Le type par défaut pour les valeurs *réelles*
  - Si on écrit `4.0`, le compilateur “crée” une valeur constante de type `double`
  - Les *réels* sont approximés par une représentation en “virgule flottante” (*floating-point*)

Les constantes

- On aimerait pouvoir utiliser un identifiant plutôt que de répéter la même valeur dans le code
- Définir une constante :

```
const double pi = 3.141592;
```

Mot clé | Type | Nom | Valeur (obligatoire!)

Les constantes

- Définir une constante :

  `const type nom = valeur`

- La valeur peut être une expression
- On ne peut pas modifier une constante 🙄
- On ne peut pas lui donner n'importe quel nom
  - Ne peut pas commencer par un chiffre
  - Il y a des identifiants réservés à éviter (`int`, `double`, etc.)

```
const int diametre_cm = 30;

const double surface_cm2 =
    pi * (diametre_cm / 2.0) *
    (diametre_cm / 2.0);

diametre_cm = 40; // Erreur!
```

Les variables

- Les variables sont utilisées pour stocker des valeurs
- On les définit, comme les constantes, mais sans le mot-clé `const`

```
int foo = 10;
```

- Pas besoin de toujours les initialiser

```
int bar;
// on ne sait pas quelle valeur s’y trouve
```

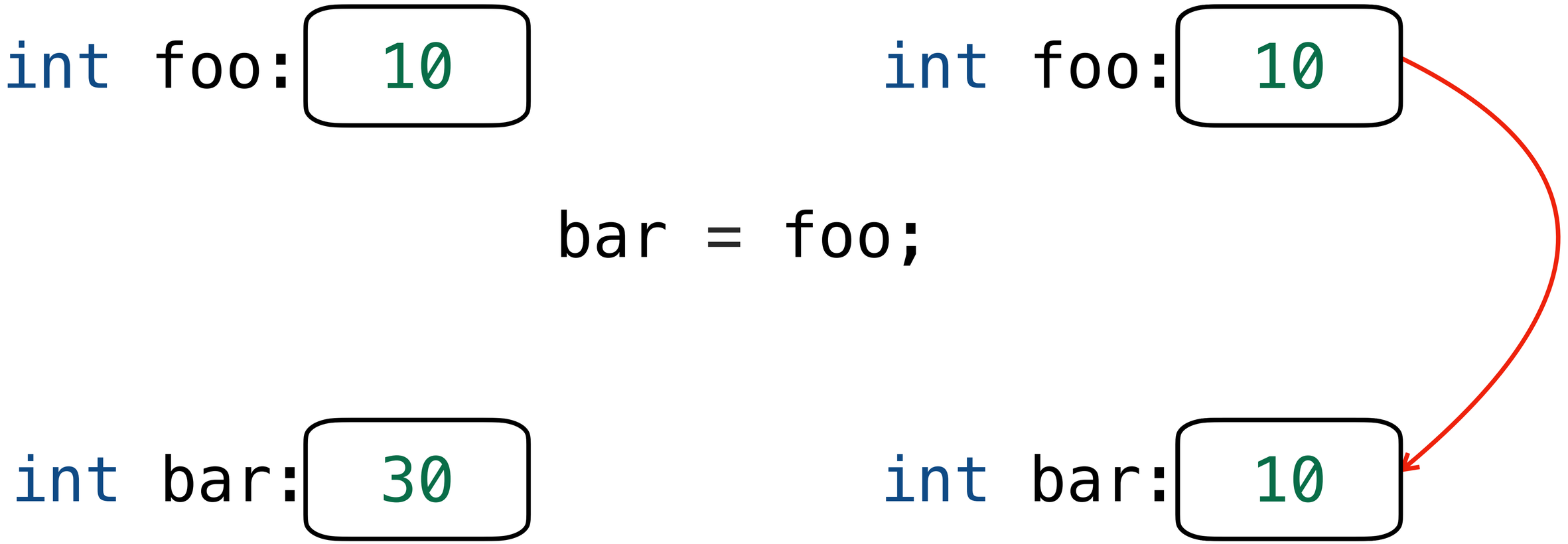
L’opérateur d’affectation “=”

Assignment operator

- Stocke une nouvelle valeur dans la variable
- Ça peut être le contenu d’une autre variable

```
int foo = 10;
int bar; //non initialisée

bar = 30;
// bar a reçu la valeur 30

bar = foo;
// maintenant bar vaut 10

foo = 50;
// et foo vaut 50
```

Affectation



Tableaux

Arrays

- Un tableau stocke plusieurs valeurs du même type

  `type nom[taille_max] = valeur`

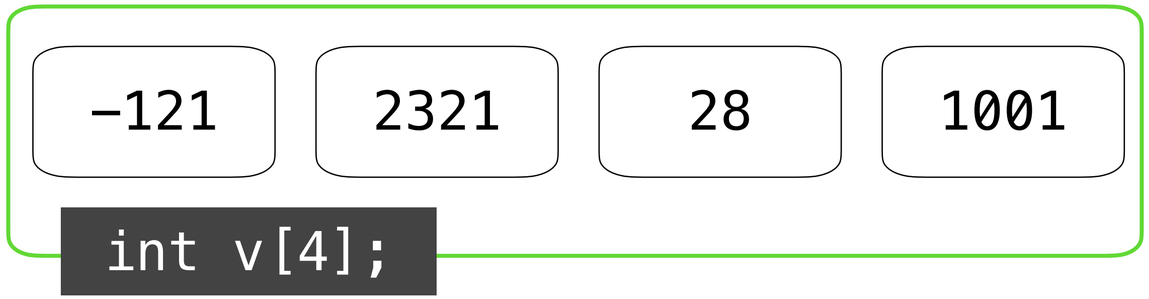
- On initialise le tableau soit en fournissant les valeurs à la définition…
- … soit en affectant des valeurs à chaque élément un par un à l'aide de l'opérateur d'indice `[]` (*subscript*)
- En C (et pas que) les indices commencent à 0
- … et vont jusqu'à `(taille_max - 1)` ⚠️

```c
// temperatures moyennes par mois
double temperatures[12] = {
    -1.2, 1.0, 4.2, 6.5,
    13.4, 16.7, 18.5, 17.4,
    11.5, 11.7, 4.6, 0.5};

temperatures[7] = 21;
temperatures[0] = -2.7;

temperatures[12] = 3; // (warning)
```

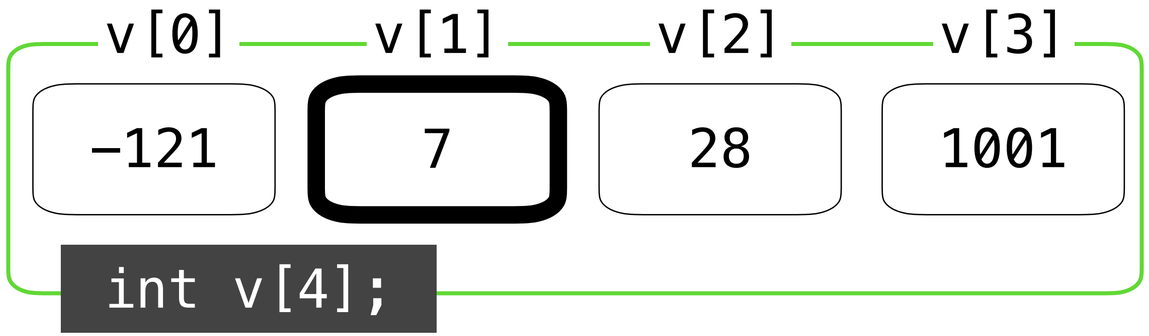
```
warning:
array index 12 is past the end of the array
(which contains 12 elements) [-Warray-bounds]
```

Tableaux

Arrays




```
v[1] = 7;
```



Caractères

- `char =` type d'un *caractère*
- Guillemets simples (*single-quotes*) `'a'`, `'b'`
- Caractères spéciaux - notés par `\` (*back-slash*) suivi d'une lettre ou d'un chiffre
  - Retour à la ligne `'\n'` (*new line*)
  - Tab `'\t'`

Chaînes de caractères

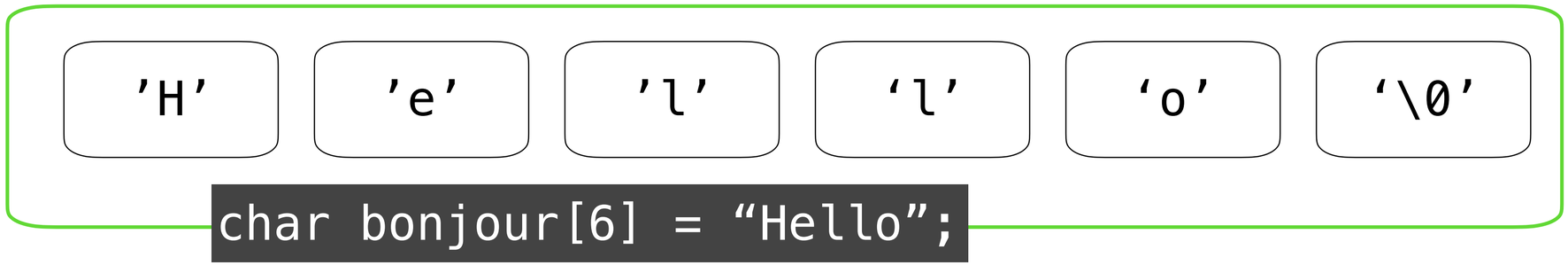
Strings

- `char[]` = *chaîne de caractères* (*string*)
- Si on écrit `"Hello"` (*double-quotes*) le compilateur crée un tableau de 6 caractères
  - Le 6e caractère = caractère spécial qui marque la fin `'\0'`
- Malheureusement on ne peut pas simplement affecter un nouveau *string* à une variable de type `char[]`
- Nous pouvons modifier chaque caractère

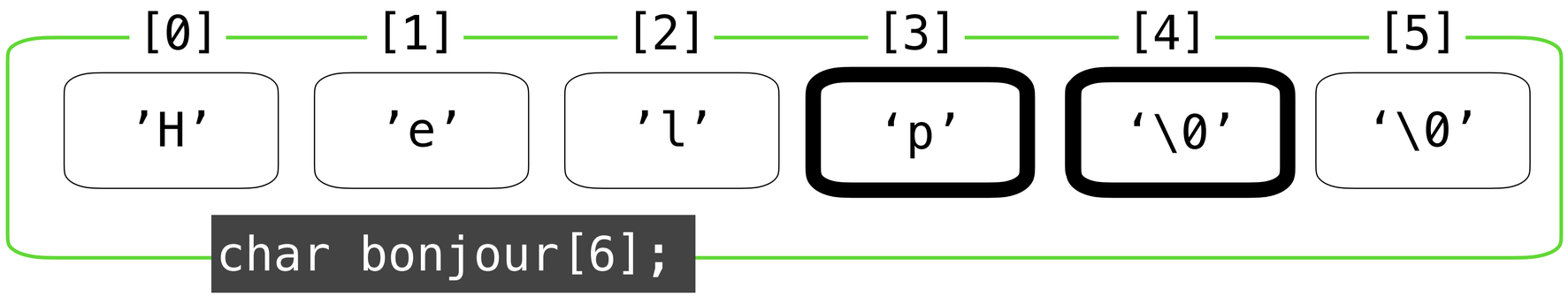
```
char bonjour[6] = "Hello";

bonjour = "Quoi?"; // Erreur!

bonjour[3] = 'p';
bonjour[4] = '\0'; // end of string
// bonjour contient "Help"
```

Chaînes de caractères

Strings




```
bonjour[3] = 'p';
bonjour[4] = '\0';
```



Afficher

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

DCT, 2023.12

Interaction avec l'utilisateur

- Le programme peut afficher du texte à la sortie standard (*standard output*, *stdout*)
- Des fonctions IO (*input-output*) sont définies dans une bibliothèque pré-installée `libc`
- Par exemple, la fonction `printf`
- Il faut inclure les déclarations de ces fonctions

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

Afficher avec printf

- On sait comment afficher un string
- En général, on aimerait afficher aussi des valeurs stockées dans des variables

```
printf(<format_string>, valeur1, valeur2, ...)
```

Chaîne de format

Format string

- On doit d'abord construire un *format string* qui décrit ce qu'on veut afficher

```
"Voici un entier %d et puis un réel %g"
```

- Contient des marqueurs `%` où on aimerait insérer des valeurs
- Le marqueur est suivi de caractères qui indiquent le type de la valeur

`%s`	`string`
`%c`	`char`
`%g`	`double`
`%d`	`int`
`%lf`	`double`

Affichage avec `printf`

```c
printf("Voici un entier %d et puis un réel %g", 13, 3.14);

// Affiche:
// Voici un entier 13 et puis un réel 3.14
```