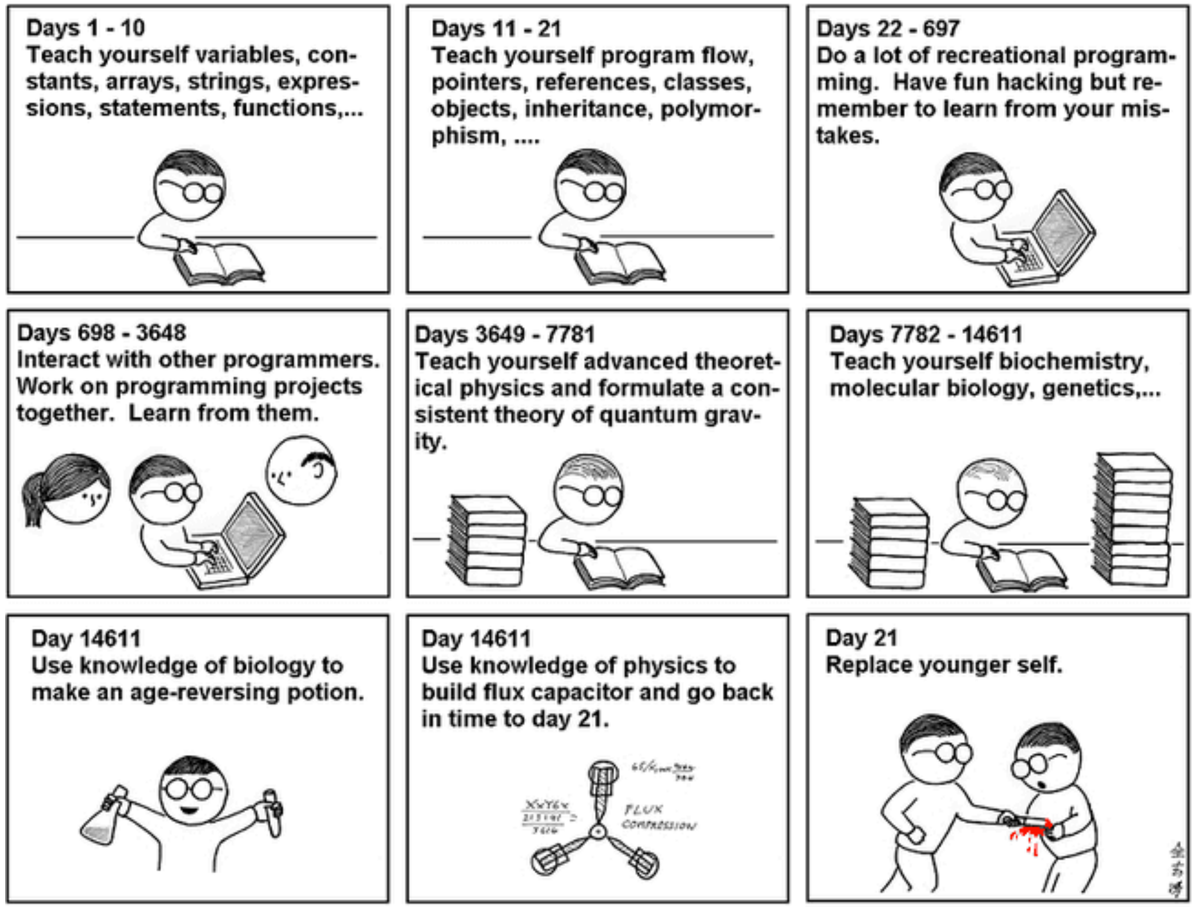
Le langage C

ICC: Cours 1



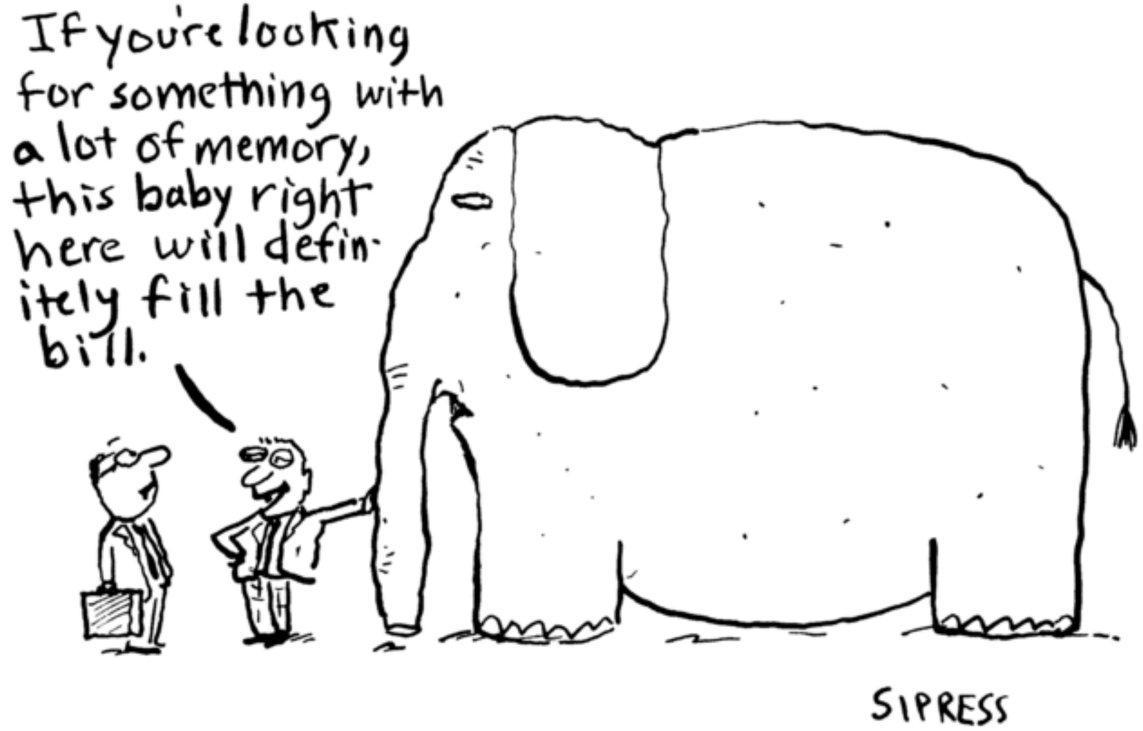
https://abstrusegoose.com/249

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*)

Pourquoi C?

- Les instructions que le CPU comprend sont très basiques - code machine
- Le programme ci-contre écrit en assembleur (x86-64) affiche à l'écran:

```
Hello, world!
```

```
global _start

section .text

_start:
  mov rax, 1
  mov rdi, 1
  mov rsi, msg
  mov rdx, msglen
  syscall

  mov rax, 60
  mov rdi, 0
  syscall

section .rodata
  msg: db "Hello, world!", 10
  msglen: equ $ - msg
```

https://jameshfisher.com/2018/03/10/linux-assembly-hello-world/

Pourquoi C?

- Des langages “haut-niveau” sont nécessaires!
- **1970**: *Dennis Ritchie* crée le langage C
- **1973**: Le système d’exploitation Unix est réécrit en C
- Python est écrit en C
  - 20.02.1991
    🎉 Joyeux 33e anniversaire! 🎂
- **Aujourd’hui**: surtout utilisé dans les systèmes d’exploitation + programmation embarquée

Hello, World!

- Beaucoup plus concis que le code assembleur
- Pourtant assez bas-niveau et puissant!

```c
#include <stdio.h>

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

Compilation

- Un programme C est défini dans *plusieurs* fichiers texte (extensions .c et .h) qui forment le code source (*source code*)
- Le code C doit quand même être traduit en code machine
- Cette “traduction” s’appelle compilation
- Compilateurs
  - gcc, clang, …

Code source

- Le code source est composé de déclarations et définitions de fonctions (e.g., `main, printf`)
- Chaque définition de fonction contient une suite d'instructions
- Chaque *programme* a une fonction spéciale dite "d'entrée" (*entry point*) — exécutée quand on lance le programme: `main()`
- Quand on exécute une fonction les instructions sont exécutées dans l'ordre

```c
#include <stdio.h>

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

Let’s do it!


```
❯ cat hello.c                                                    = “Affiche-moi le contenu
                                                                (texte) du fichier ‘hello.c’”
-------------------------------------------------------------------------------------------------
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    printf("Bonjour, ICC!\n");
}
-------------------------------------------------------------------------------------------------
❯ gcc -o hello hello.c                                           = “Compile le fichier ‘hello.c’
                                                                et crée l’executable ‘hello’”
-------------------------------------------------------------------------------------------------
❯ ./hello                                                        = “Lance le programme ‘hello’”
-------------------------------------------------------------------------------------------------
Hello, World!
Bonjour, ICC!
-------------------------------------------------------------------------------------------------
❯ ls -lh hello*                                                  = “Décris les fichiers qui se
-rwxr-xr-x  1 dntz  staff    48K Dec 27 17:47 hello             trouvent dans le répertoire
-rw-r--r--  1 dntz  staff    97B Dec 27 17:46 hello.c           courant dont le nom commence par
                                                                ‘hello’”
```

Bon à savoir

Commentaires

```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 ligne
    printf("Bonjour, ICC!\n"); // Je suis un autre commentaire sur une ligne
}
```

Bon à savoir

Formatage

- Le compilateur s'intéresse à la syntaxe, pas au formatage du code (ce n'est pas du Python…)
- Cependant, pour que le code soit lisible, c'est mieux de ***toujours*** utiliser un formateur
  - Dans VS Code
    - Windows: Alt+Shift+F
    - Linux: Alt+Shift+I
    - macOS: ⌥+⇧+F
- tabs vs. spaces https://youtu.be/SsoOG6ZeyUI

Ceci est un programme C valide!

IOCCC 2011

https://www.ioccc.org/years.html#2011_akari

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 nous aident à éviter des ***erreurs de conception***!
  - Une erreur pendant la compilation est infiniment préférable à une erreur pendant l’exécution
  - Les langages qui “attrapent” le plus d’erreurs pendant la compilation sont les langages fonctionnels — Scala, Haskell, OCaml, F#, …

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 type nom = valeur`
- La valeur peut être une expression
- On ne peut pas modifier la valeur d'une constante
- Le nom ne peut pas commencer par un chiffre
- Il y a des identifiants réservés à éviter (`int`, `double`, etc.)

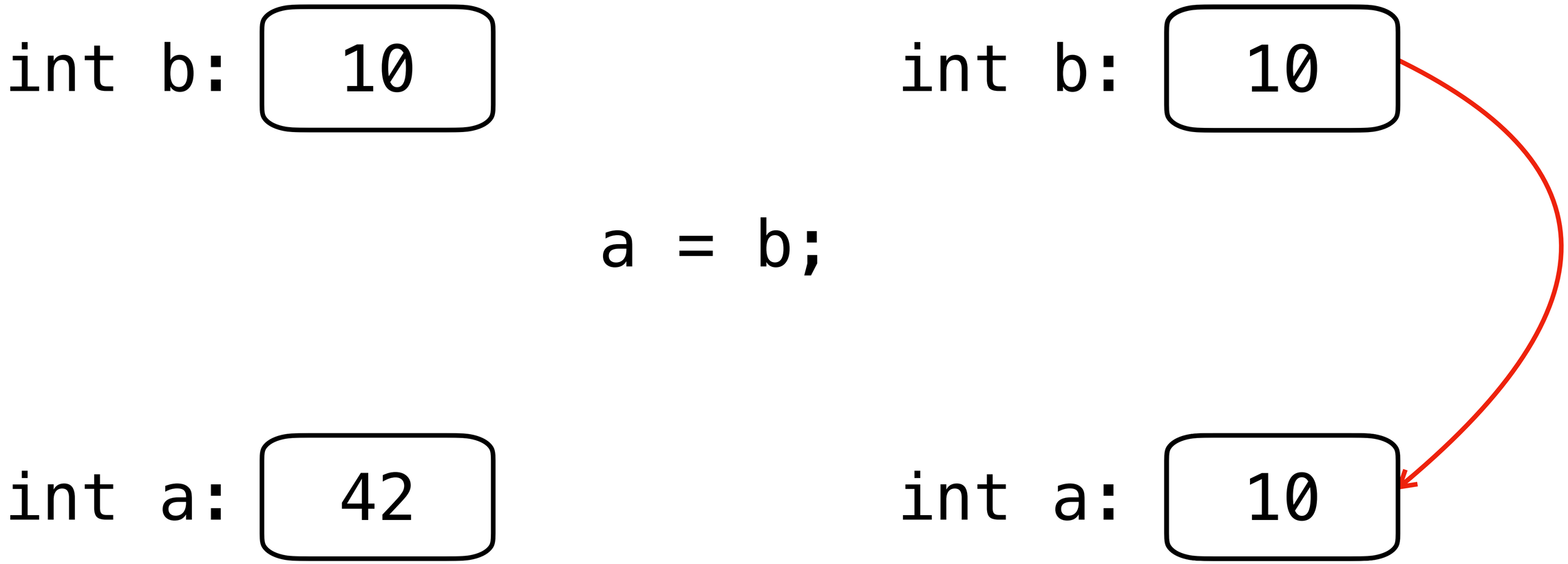
```
const double pi = 3.141592;
const int diametre_cm = 30;

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`
- On peut les initialiser (ou pas!)
- L'opérateur d'affectation (*assignment operator*) "=" stocke une nouvelle valeur dans la variable
- E.g., le contenu d'une autre variable

```
int ma_variable = 10;
int mon_autre_variable; //non initialisée

mon_autre_variable = 30;
// mon_autre_variable a reçu la valeur 30

mon_autre_variable = ma_variable;
// maintenant mon_autre_variable vaut 10

ma_variable = 50;
// et ma_variable vaut 50
```

Bon à retenir



Tableaux

Arrays

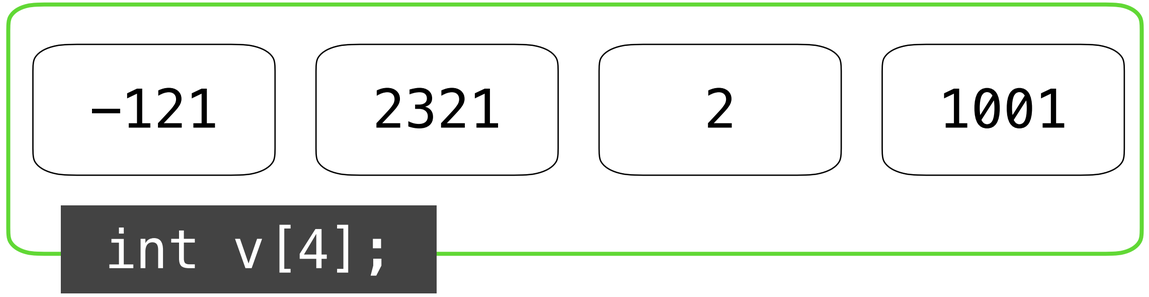
- Un tableau stocke plusieurs valeurs du même type
- `type nom[taille_max] = valeur`
- On initialise le tableau soit en fournissant les valeurs à la définition…
- … soit en affectant des valeurs à chaque élément un par un à l'aide de l'opérateur d'indice `[]` (*subscript*)
- En C (et pas que) les indices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

```c
// temperatures mensuelles
// des 5 dernières années
double temperatures[60];

// age des étudiants GM en 2044
int age_en_2044[350];

// temperatures mensuelles de 2022
double temperatures_2022[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[12] = -1.5;
temperatures[0] = -2.7;

age_en_2044[349] = 40;
age_en_2044[350] = 41; // (warning)
```

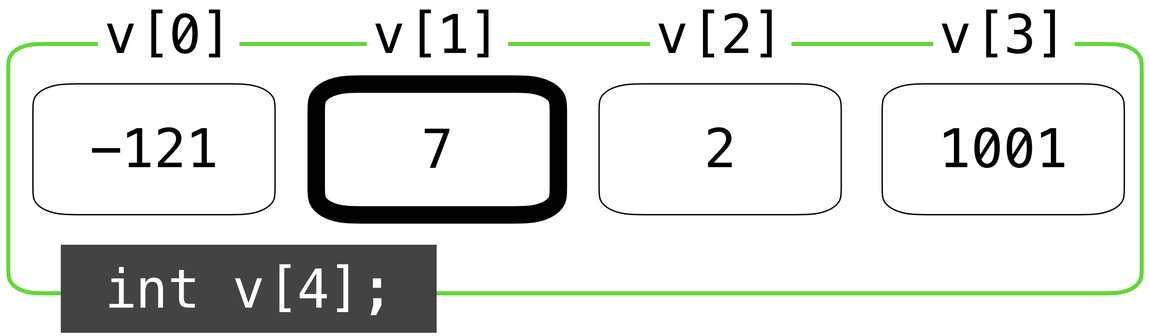
```
warning: array index 350 is past the end of the
array (which contains 350 elements) [-Warray-bounds]
```

Tableaux

Arrays




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



Chaînes de caractères

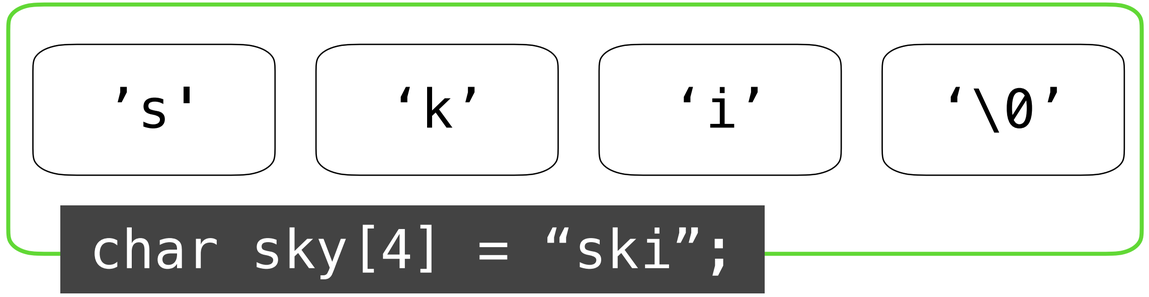
Strings

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

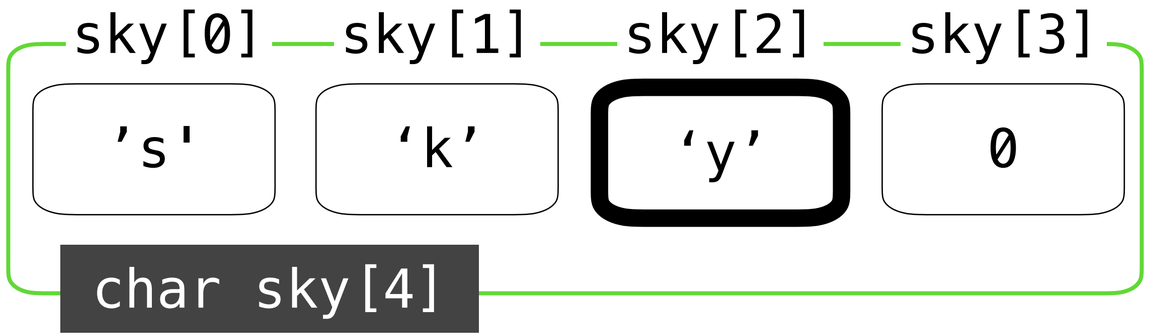
```
// Une note américaine
char grade = 'A'; // well done!

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

bonjour = "Quoi?"; // Erreur!

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

Chaînes de caractères

Strings




```
sky[2] = 'y';
```



Appels de fonctions

Function calls

- Chaque programme a une fonction `main`
- Il y a plein d'autres fonctions définies *ailleurs* qu'on peut utiliser (par ex. `printf`)
  - C'est des bibliothèques (*libraries*) qu'on peut installer!
  - On réutilise le code écrit par d'autres développeurs
- Une fonction mathématique a un domaine et un co-domaine $f : \mathbb{R} \times \mathbb{Z} \to \mathbb{R}$
- Une fonction C aussi!

```
double f(double x, int y); // f: double, int -> double
```

- Déclaration de la fonction

Appels de fonctions

Function calls

- `x` et `y` sont les paramètres de la fonction `f`
- la fonction `f` retourne une valeur de type `double`
- `u` est le paramètre de la fonction `g`
- la fonction `g` retourne `int`
- Quand on utilise une fonction on dit qu'on l'appelle (*calling a function*)
- Les valeurs transmises à la fonction sont des arguments
- Ce n'est pas nécessaire de stocker le résultat de l'appel

```
double f(double x, int y);
// La définition est ailleurs...

int g(char u[]);

double resultat;
resultat = f(100.0, 4);
// 100.0 et 4 sont les arguments

const double pi = 3.14;
f(pi, -13); // ok
g("L'argument de g"); // ok aussi
```

Afficher

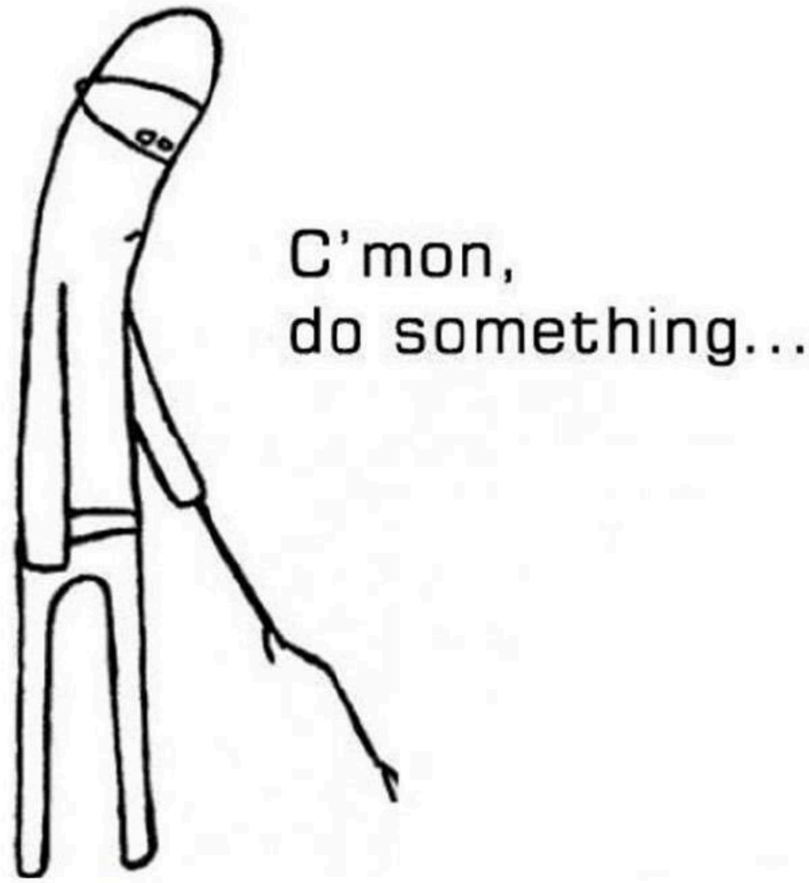
```
#include <stdio.h>
```

DCT, 2023.12

Interaction avec l’utilisateur

- C’est bien de calculer des valeurs, mais on en fait quoi?
- On aimerait bien interagir avec l’utilisateur, par exemple en affichant du texte
- Heureusement, des fonctions IO (*input-output*) sont définies dans la bibliothèque pré-installée appelée `libc`
- Il faut inclure les déclarations de ces fonctions

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



Chaîne de format

Format string

- La sortie standard (*standard output*, *stdout*) permet au programme d'afficher des *strings*
- On doit d'abord construire un *format string* qui reflète ce qu'on veut afficher

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

- Un *format string* contient des marqueurs % où on aimerait insérer des valeurs
- Le marqueur est suivi d'un autre caractère qui indique le type de la valeur

%s	string
%c	char
%g	double
%d	int
%f	double

Affichage avec `printf`

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

- La fonction `printf` met en forme le résultat désiré et l'affiche au *stdout*
- le premier paramètre est une chaîne de format (*format string*)
- les paramètres successifs correspondent aux valeurs qu'on veut afficher

```c
const double pi = 3.14159265358979323846;
const int diametre_cm = 30;
const char nom[] = "Margherita";
const char score = 'B';

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

printf("La surface de %s (score %c) = %g cm²\n",
        nom,
        score,
        surface_cm2);
// Affiche:
// La surface de Margherita (score B) = 706.858 cm²
```

Bon à savoir

- Pour spécifier un “retour à la ligne” on utilise le caractère `\n`
- Pas besoin de faire une énorme *format string* sur une seule ligne
  - Les *strings* successifs (séparés par des **espaces** ou par des **retours à la ligne** — *pas de virgule!*) sont concaténées par le compilateur

```c
printf(
    "*-----------------------------------*\n"
    "|            Bonjour  %s (%d)      |\n"
    "*-----------------------------------*\n",
    "ICC", 2024);
// Affiche:
// *-----------------------------------*
// |            Bonjour  ICC (2024)     |
// *-----------------------------------*
```