C

Rappels

Dans la brochure

https://www.esiea.fr/ms-sis/

Rappels programmation C

Le but de ce module est de faire une remise à niveau en C aux étudiants qui n'auraient pas pratiqué ce langage récemment. Toutes les thématiques seront étudies à savoir les tableaux, les pointeurs, l'allocation dynamique de mémoire, les structures, les fichiers, etc. À la fin du module, un projet est donné aux étudiants dans lequel toutes ces thématiques seront à appliquer.

N’oubliez pas…

La vérité est la première des loyautés.

Gen. Pierre de Villiers

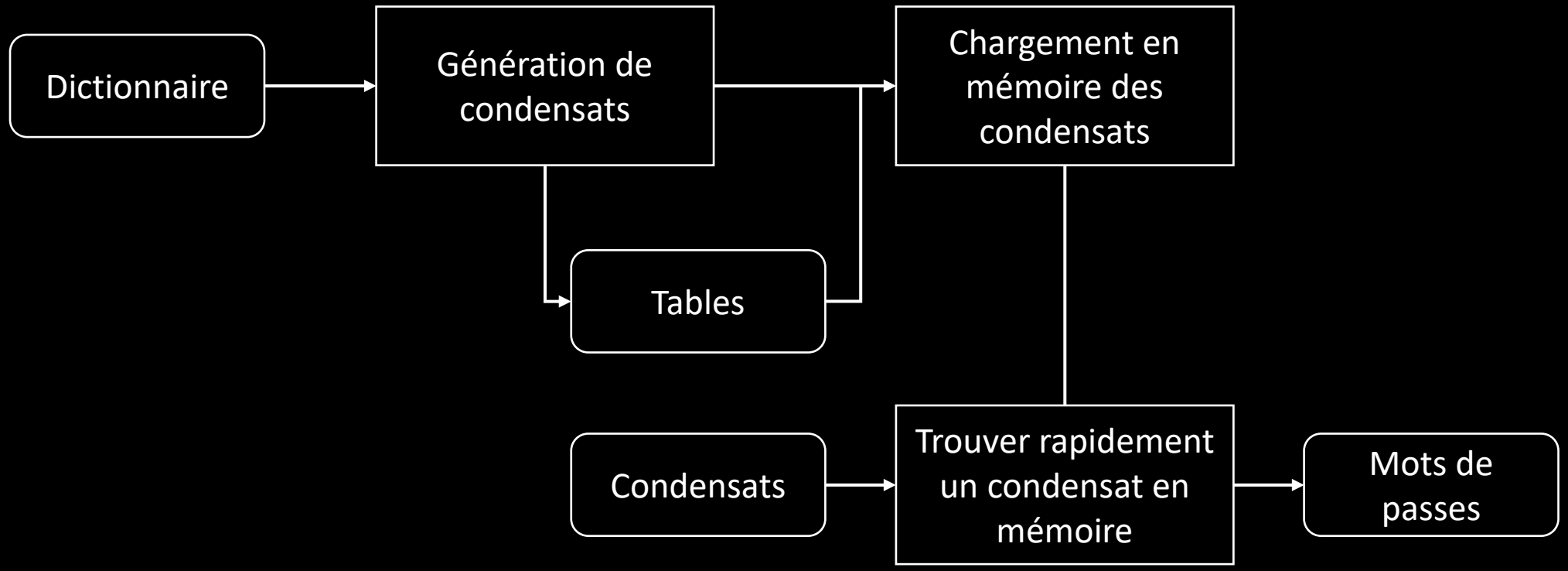
L’homme ordinaire ne se préoccupe que de passer le temps, l’homme de talent que de l’employer.

Arthur Schopenhauer

Il ne faut pas apprendre à écrire mais à voir. Écrire est une conséquence.

Antoine de Saint-Exupéry

Livrable : un repo git

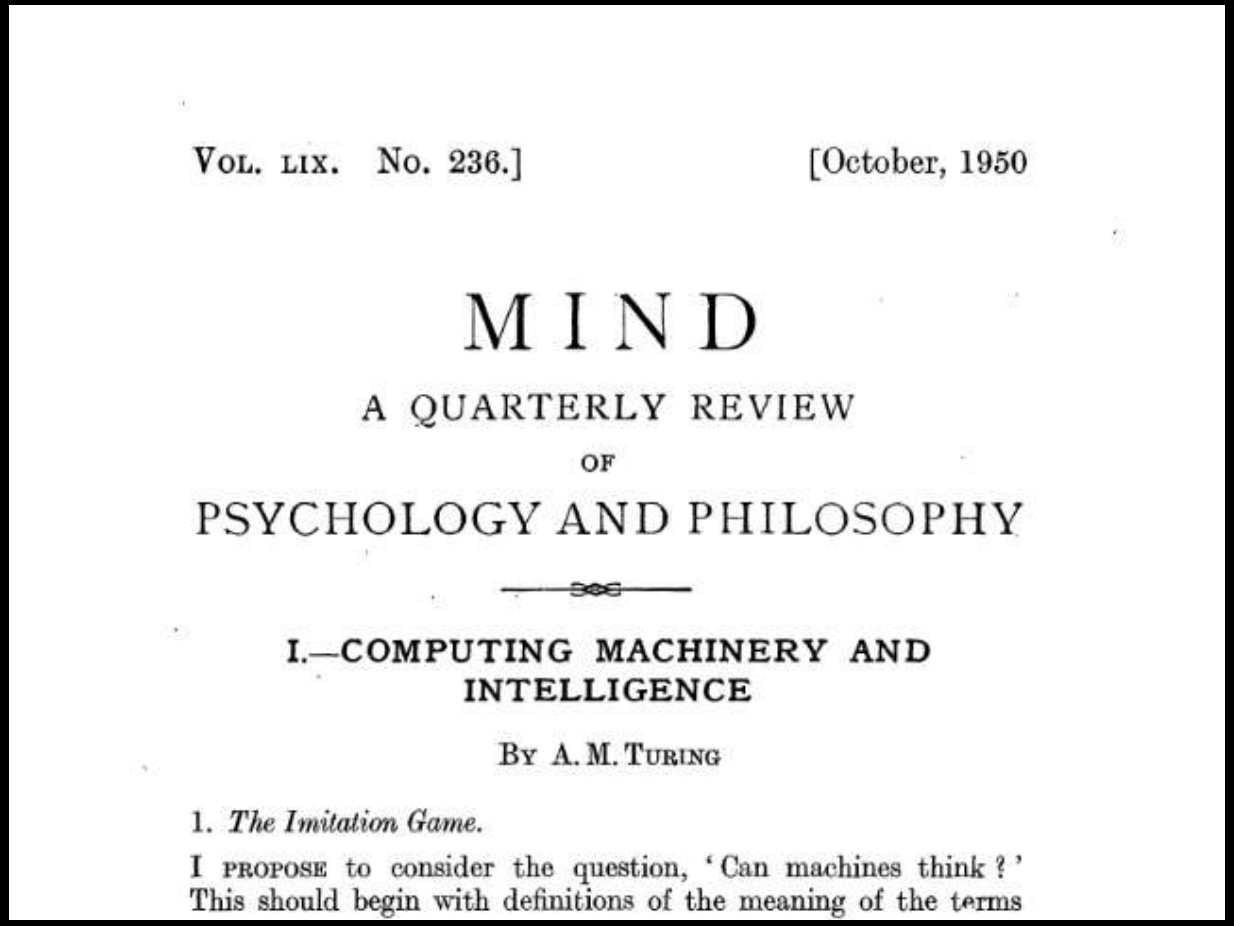


Le programme (l'autre)

- La programmation informatique
- Fonctionnement d'un ordinateur
- Compilation
- Outillage
- Rappels algorithmiques
- C
  - Entrées/sorties fichiers
  - Tableaux
  - L'objet de toutes les peurs et de nombreux scandales : les pointeurs 😈
  - Allocation et libération dynamique de mémoire
  - Structures et méta-structures

La programmation informatique

La programmation informatique



Vol. lix. No. 236.] [October, 1950

MIND

A QUARTERLY REVIEW

OF

PSYCHOLOGY AND PHILOSOPHY

I.—COMPUTING MACHINERY AND INTELLIGENCE

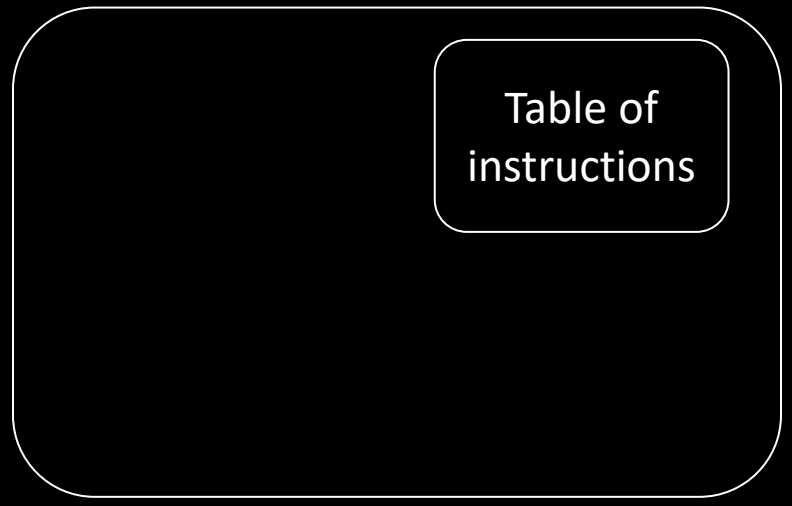
By A. M. Turing

1. *The Imitation Game.*

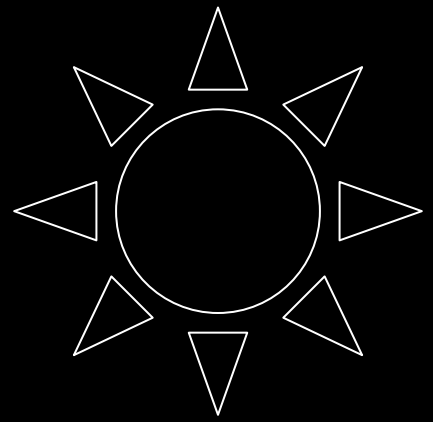
I propose to consider the question, 'Can machines think?' This should begin with definitions of the meaning of the terms

La programmation informatique

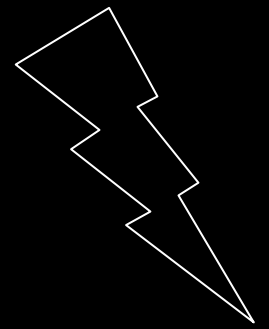
La machine de Turing



Store

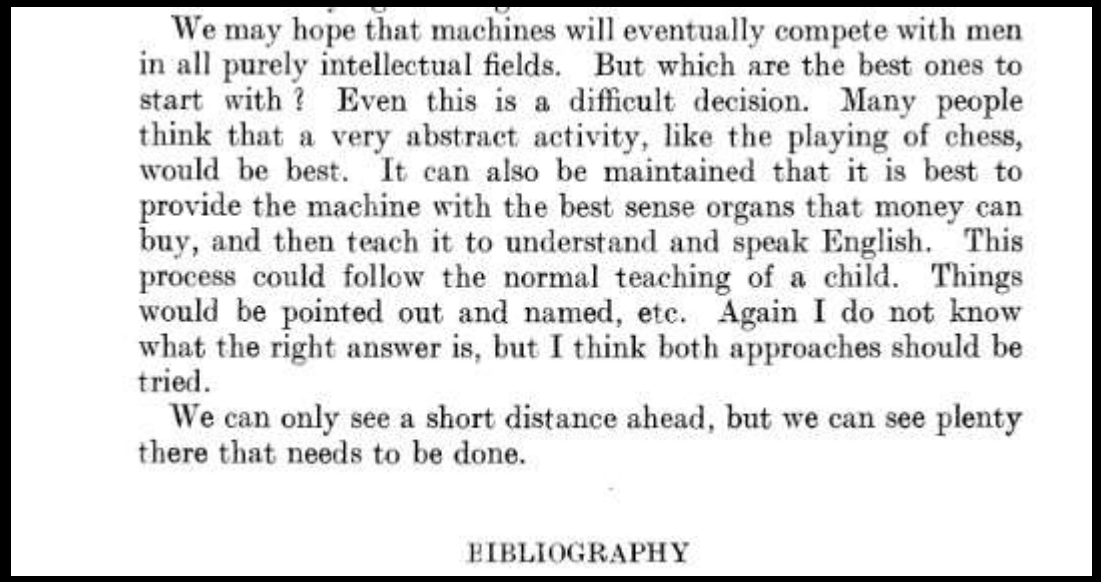


Control



Executive unit

La programmation informatique



We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc. Again I do not know what the right answer is, but I think both approaches should be tried.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

BIBLIOGRAPHY

Fonctionnement d'un ordinateur

Fonctionnement d’un ordinateur

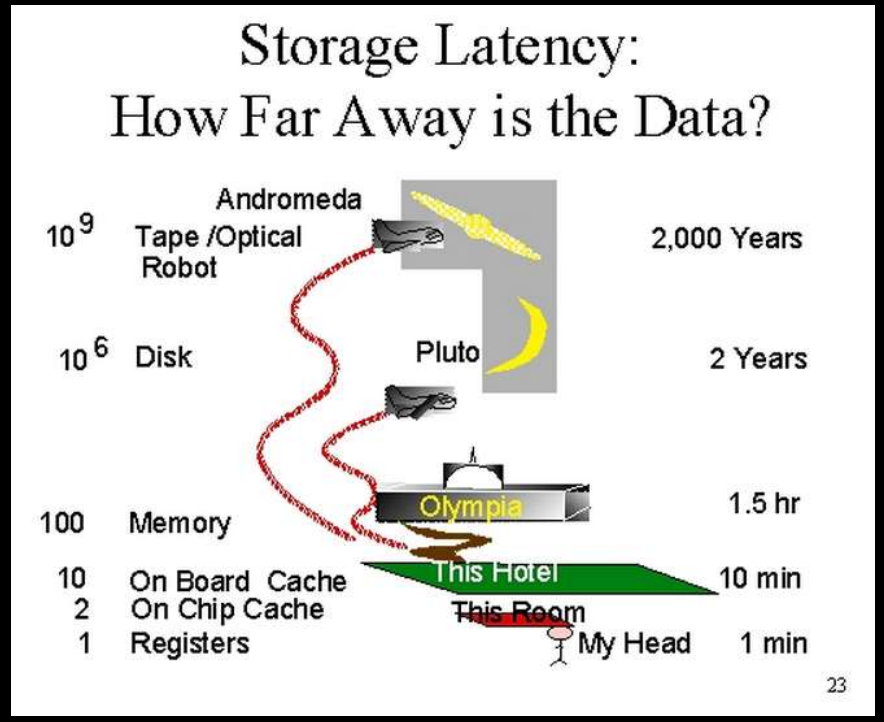
Fonctionnement d'un ordinateur

- Constitution
- Interactions
- Mémoire
- Instructions
- Pile, tas
- Noyau

« Optimisation »

https://blog.codinghorror.com/the-infinite-space-between-words/

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia



Compilation

Compilation

C (.c)

↓ Traitement par pré-processeur

C prétraité (.i)

↓ Compilation

Assembleur (.s)

↓ Assemblage

Code binaire (.o)

↓ Edition de liens

Code binaire exécutable (avec les bibliothèques)

Outillage : Docker

https://docs.docker.com/desktop/install/linux-install/

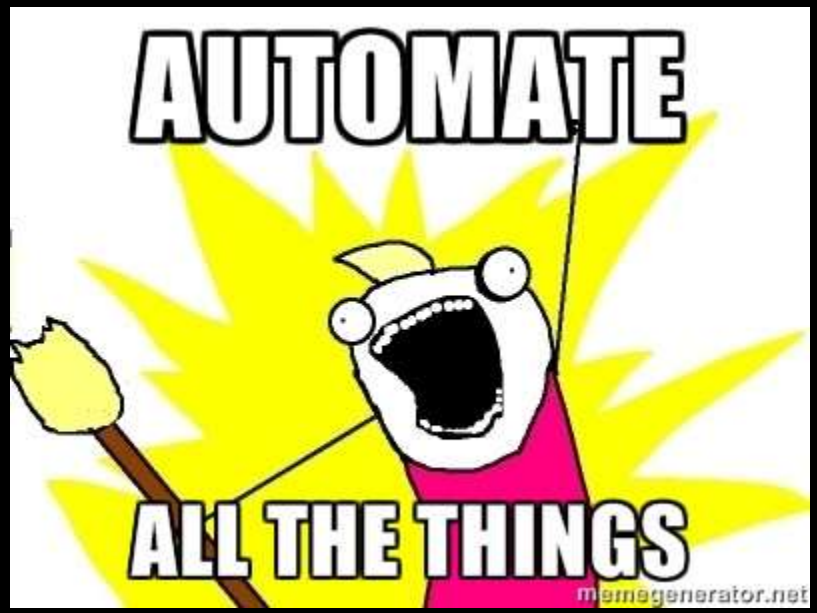
Outillage : Docker

https://hub.docker.com/_/gcc

```
$> sudo docker run --rm -ti
   -v "$(realpath .)":/usr/src/myapp
   -w /usr/src/myapp
   gcc:12 gcc -Wall hello-universe.c
```

Outillage : make

https://makefiletutorial.com/ *(premier résultat Google)*
https://scaron.info/blog/gnu-make.html



Outillage : make

```
# the compiler: gcc for C program, define as g++ for C++
CC = gcc

# compiler flags:
#  -g    adds debugging information to the executable file
#  -Wall turns on most, but not all, compiler warnings
CFLAGS  = -g -Wall

# the build target executable:
TARGET = hello-universe

all: $(TARGET)

$(TARGET): $(TARGET).c
        $(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c

clean:
        $(RM) $(TARGET)
```

https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html

Outillage : make et Docker

```
$> sudo docker run --rm -ti
    -v "$(realpath .)":/usr/src/myapp
    -w /usr/src/myapp
    gcc:12 make
```

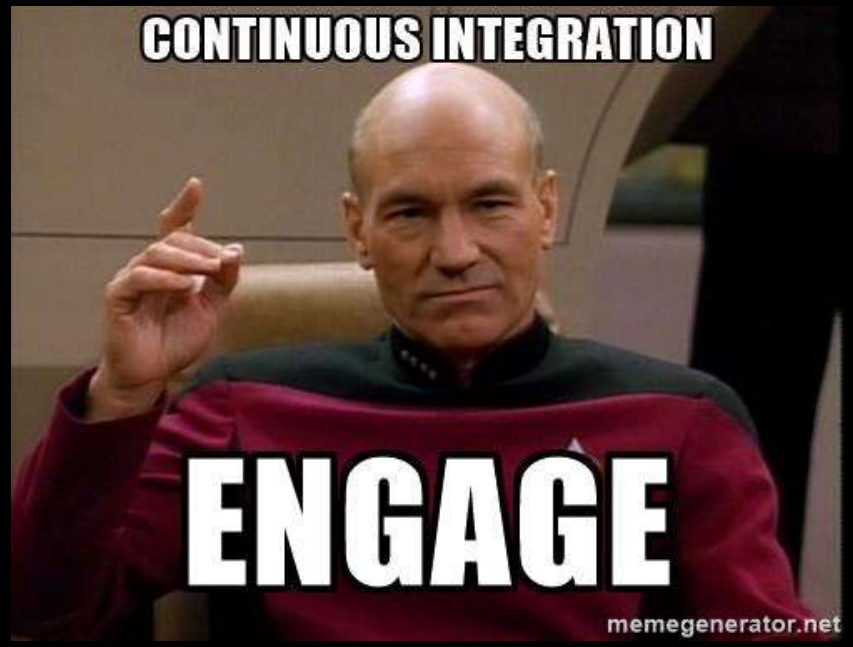
Outillage : git

https://learngitbranching.js.org/

Outillage : intégration continue

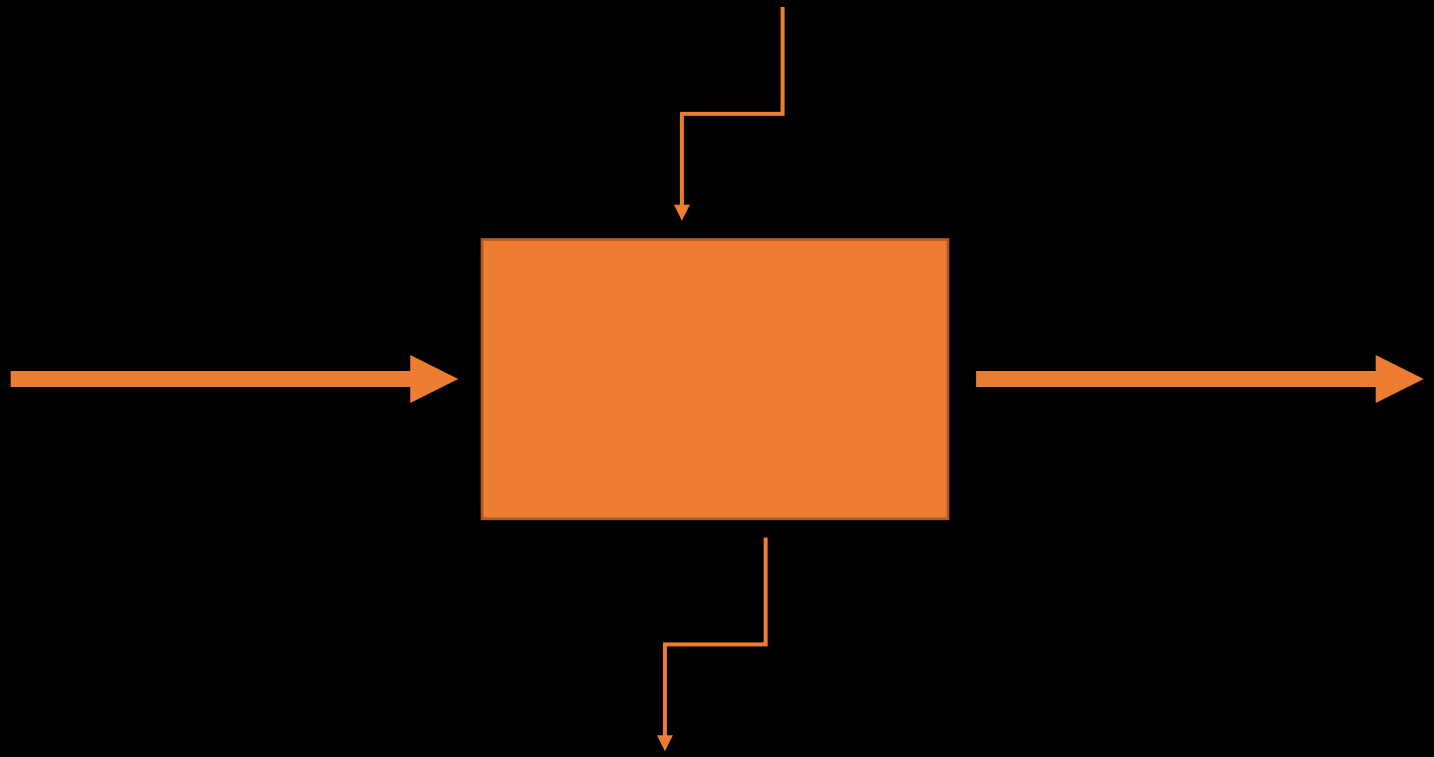
https://docs.gitlab.com/ee/ci/quick_start/

https://docs.github.com/en/actions

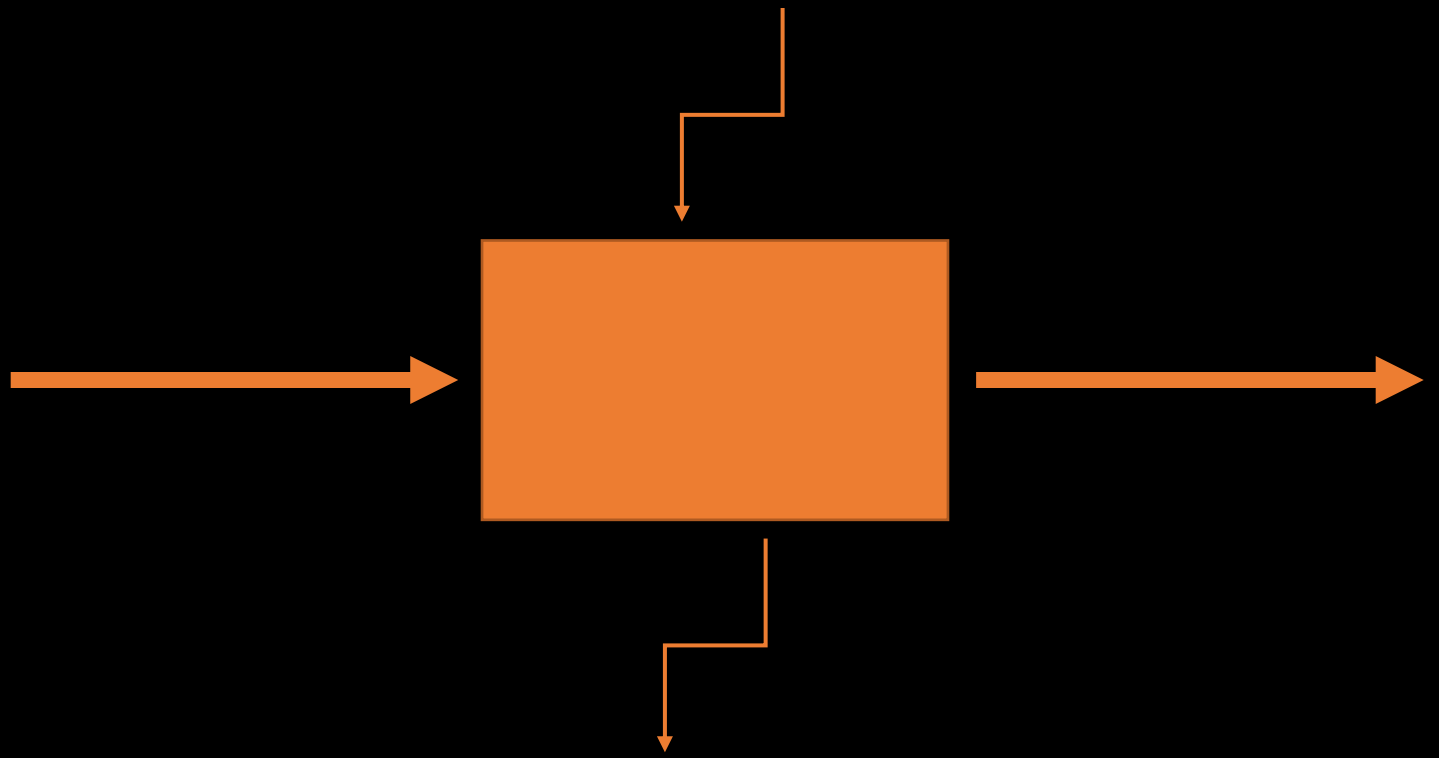


Algorithmie

Le programme

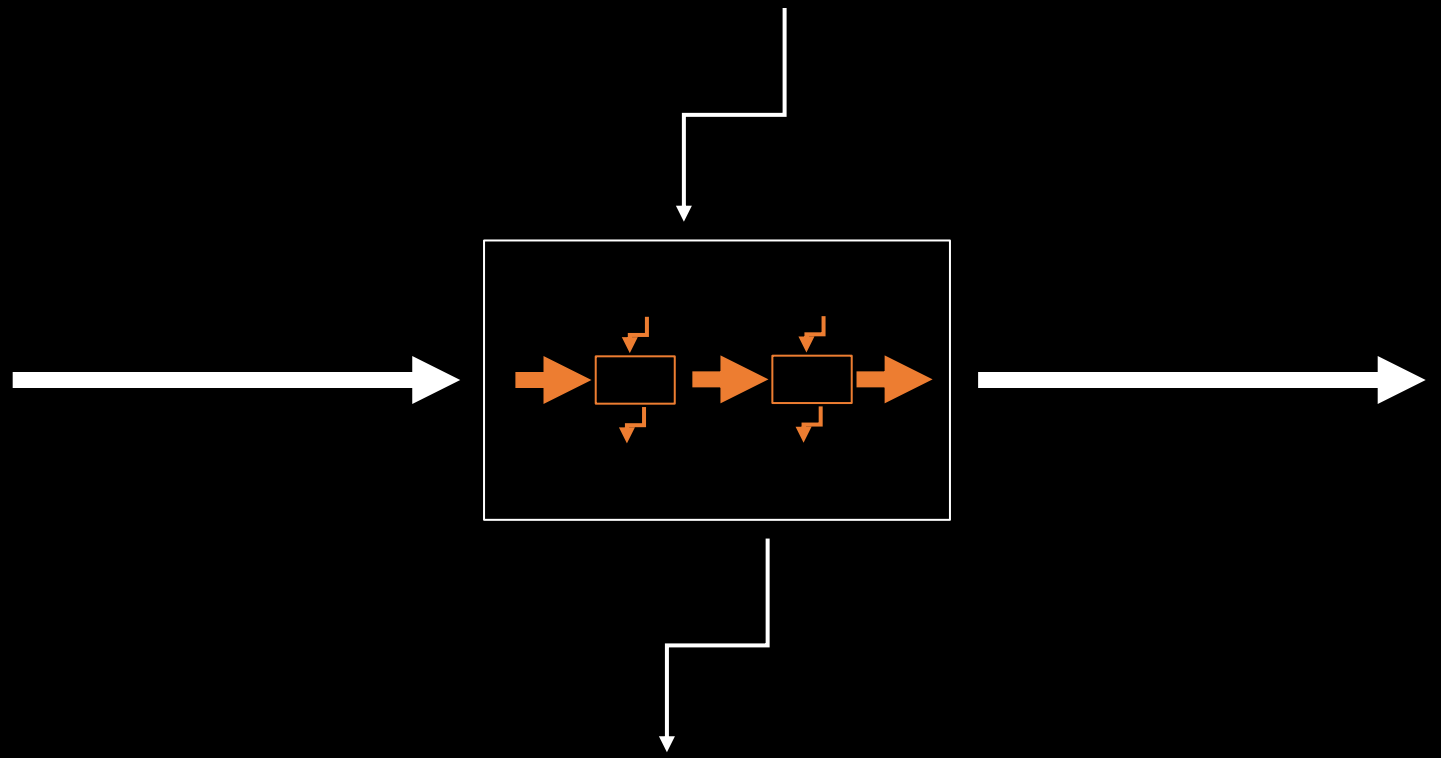


Le programme



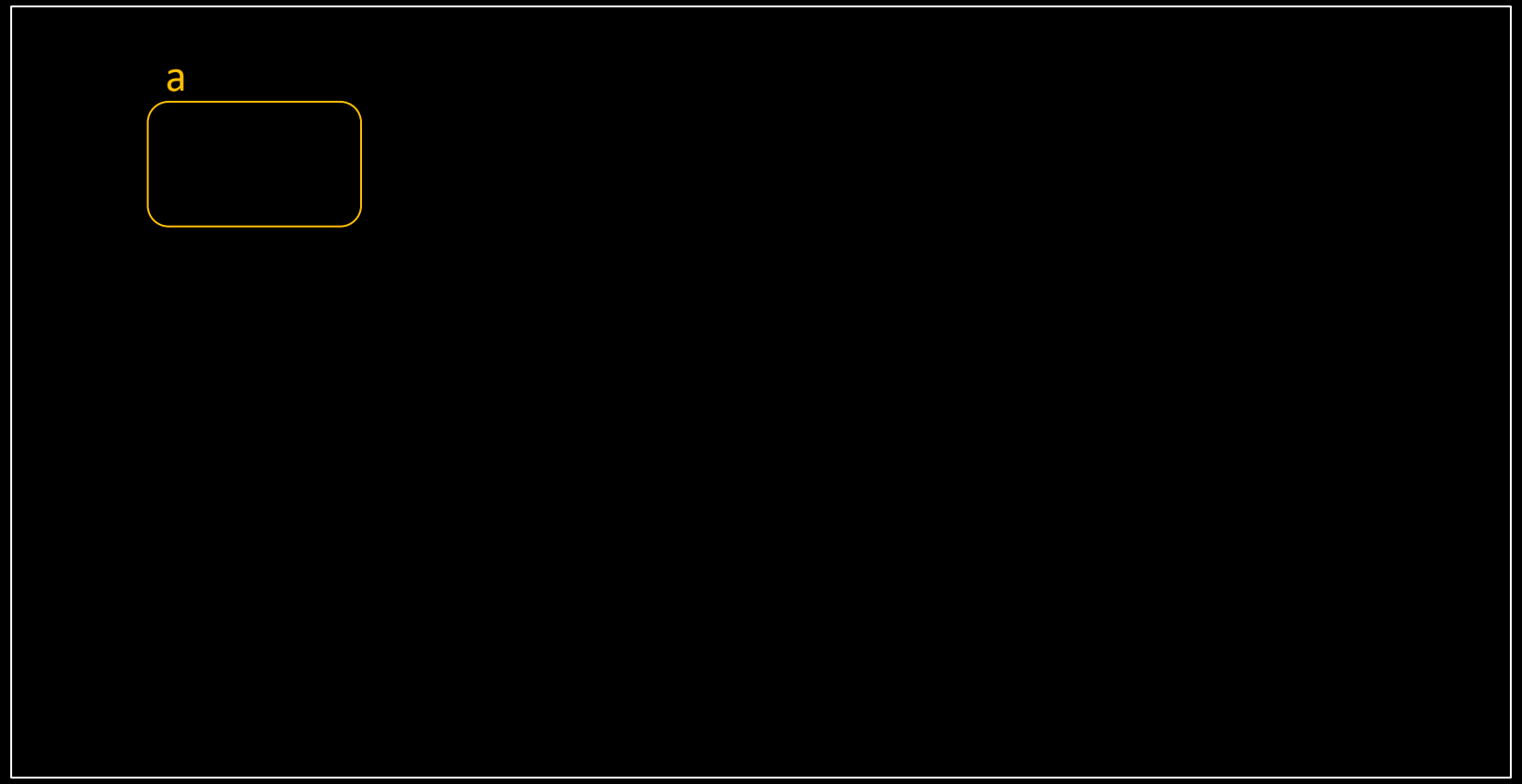
```
fonction(a, b, c, d)
```

Le (sous-)programme



```
fonction(a, b, c, d)
```

La déclaration

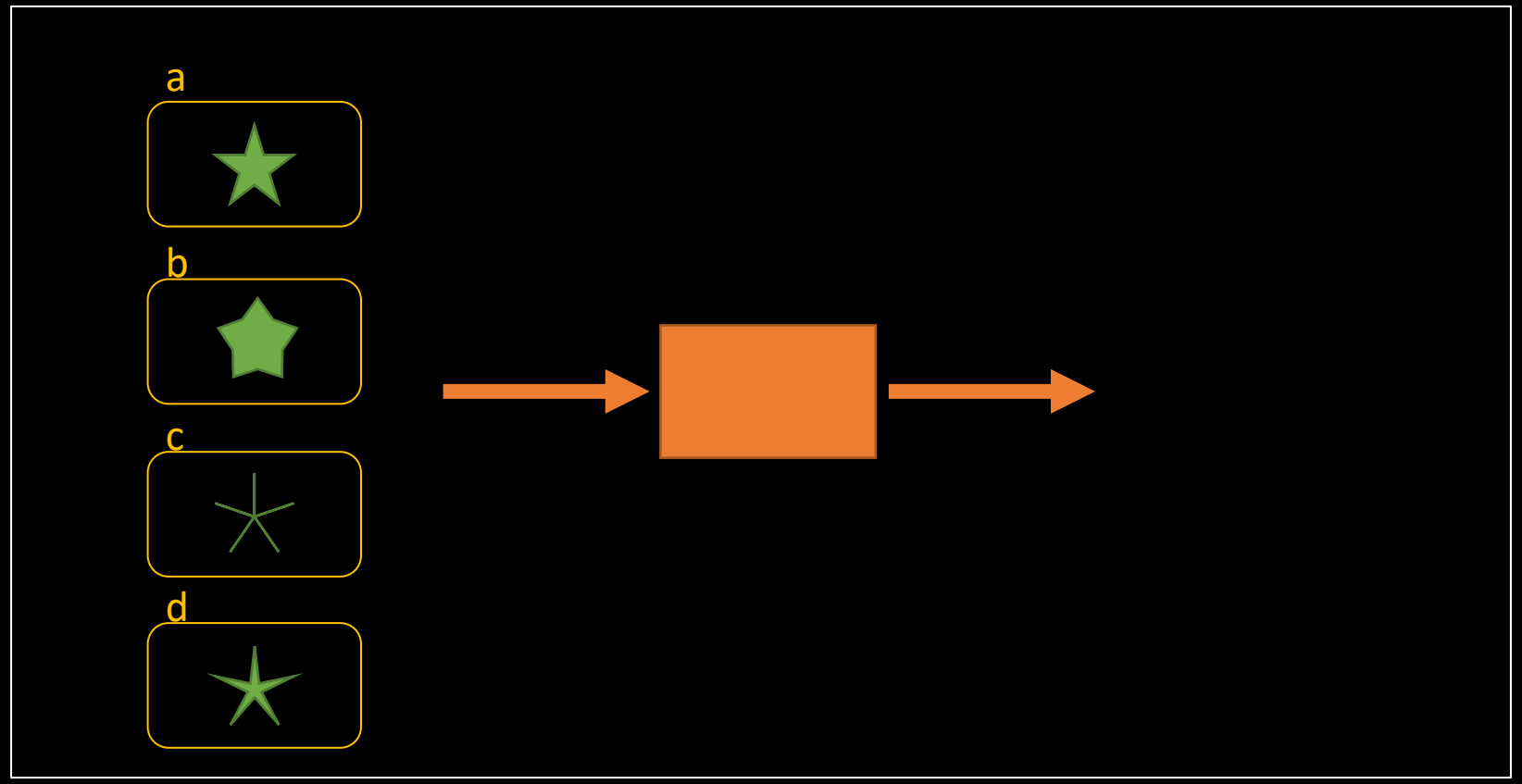



```
int a;
```

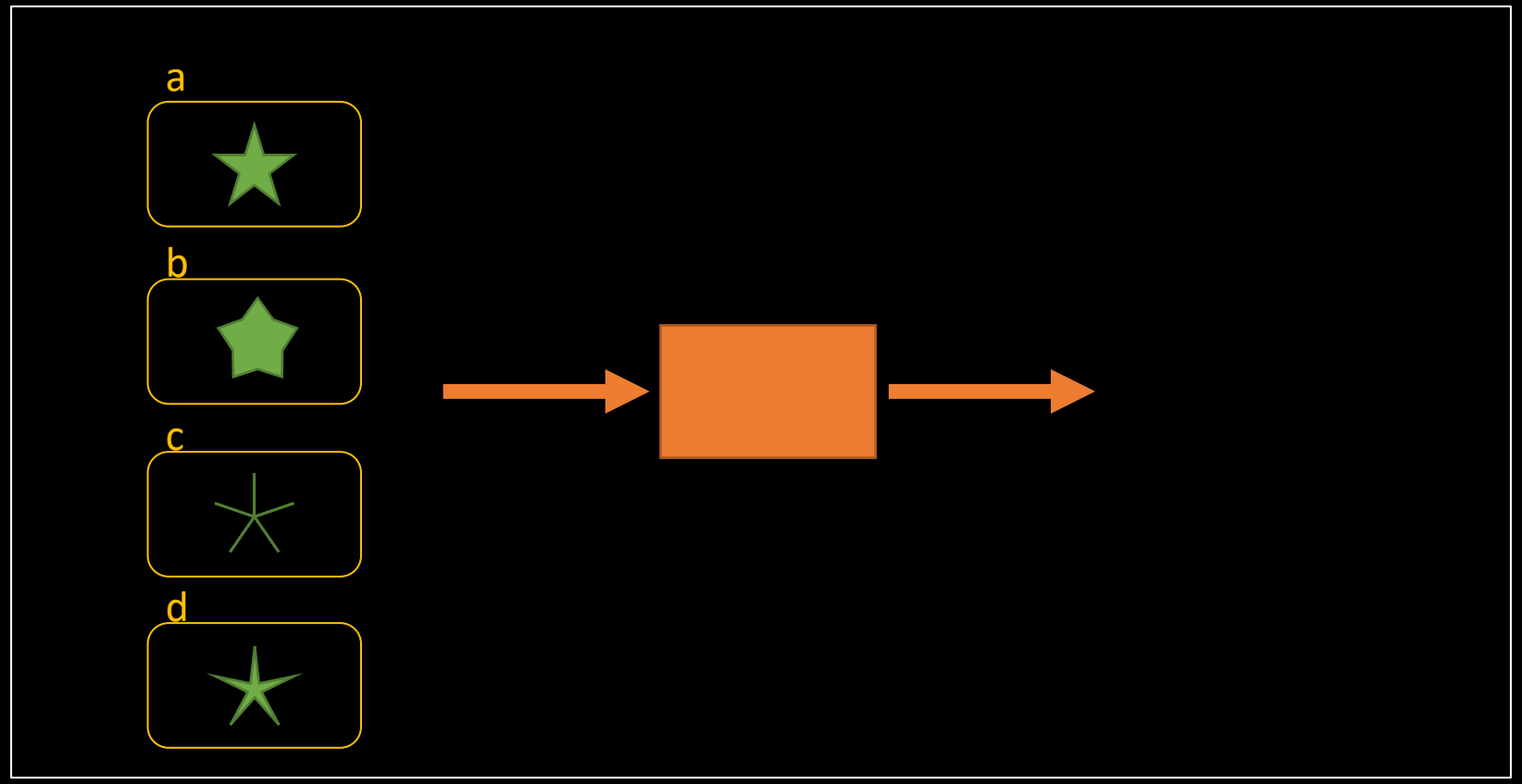
L’affectation


```
a = 3;
```

L’appel de fonction

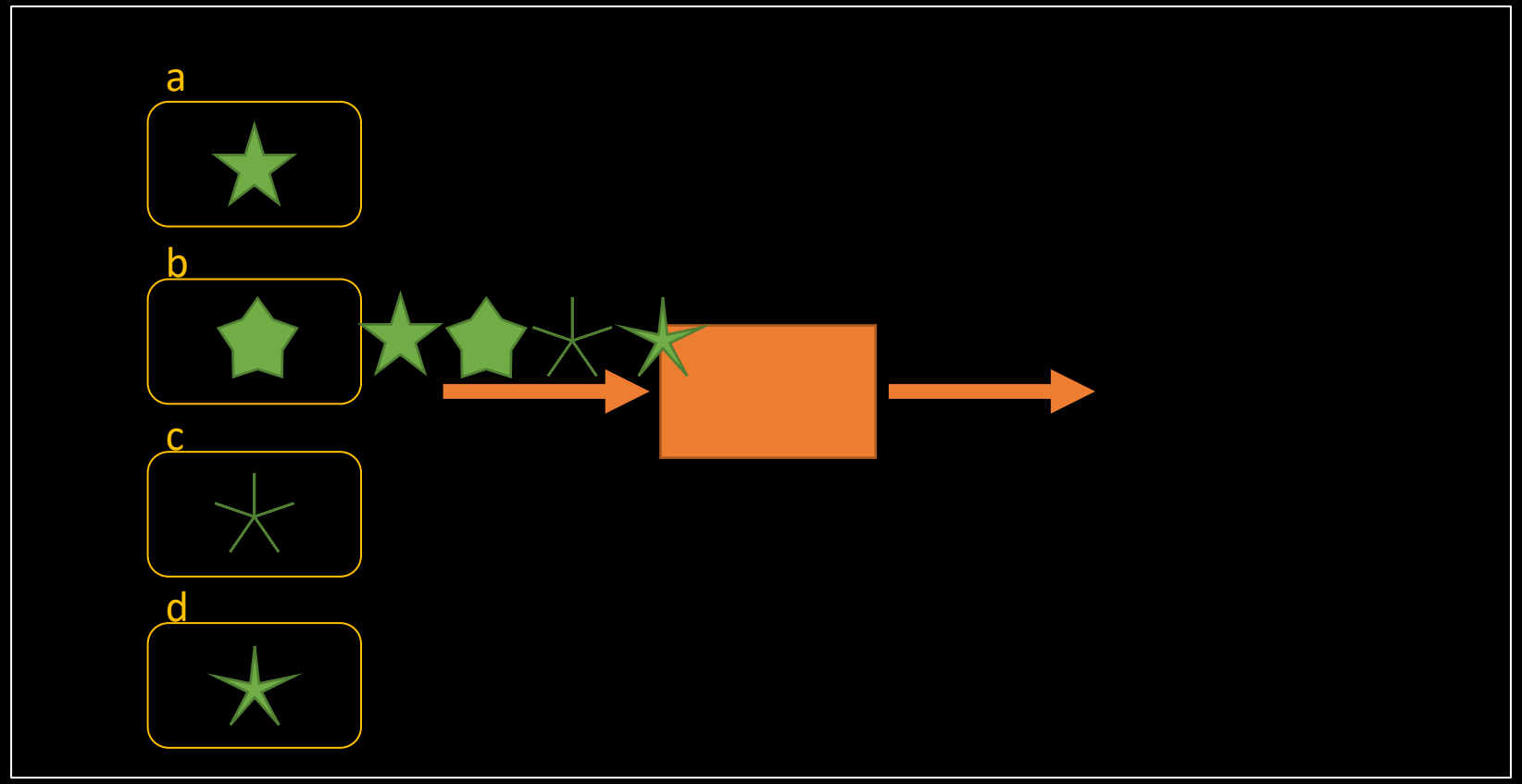


L’appel de fonction



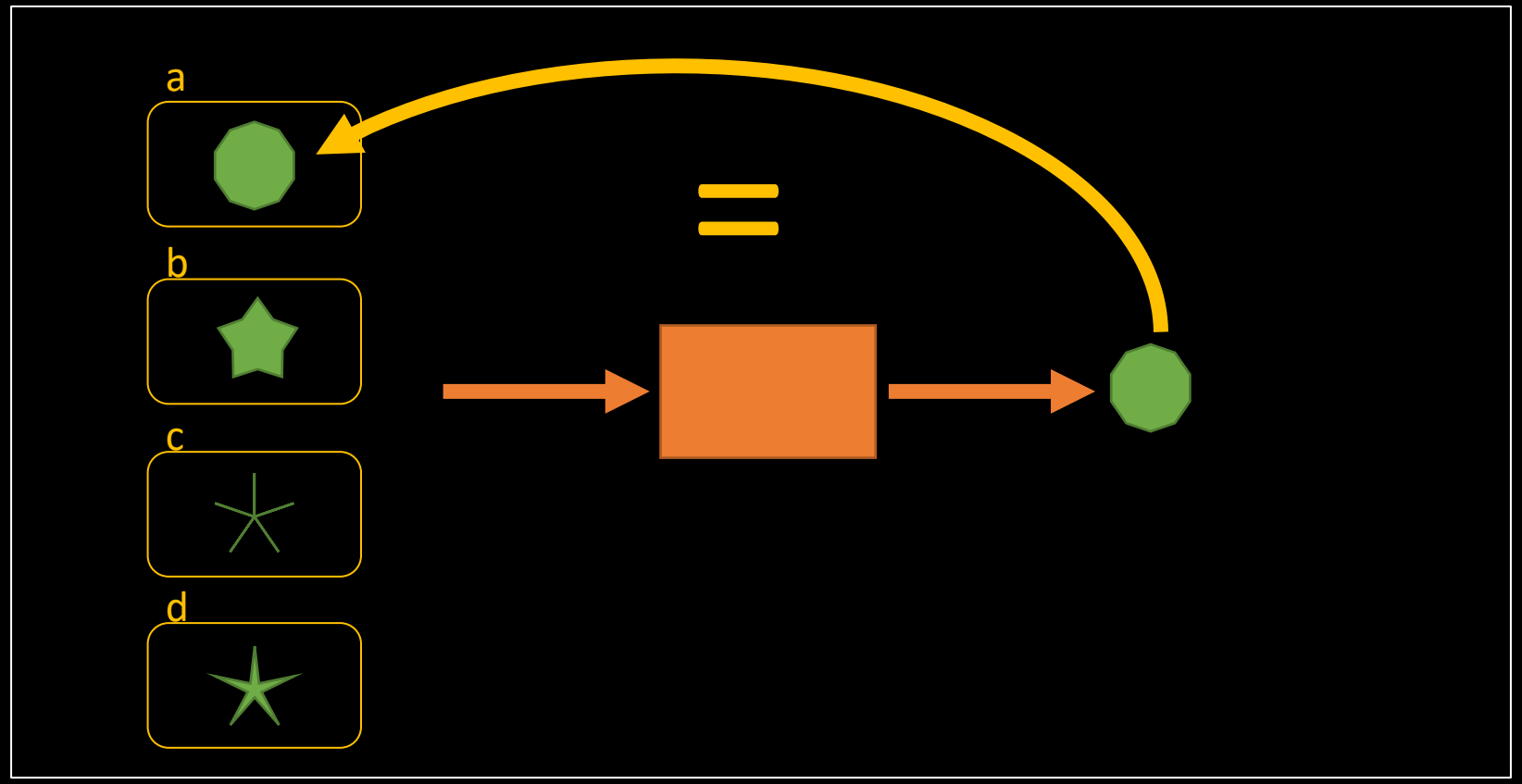

```
a = fonction(a, b, c, d)
```

L’appel de fonction



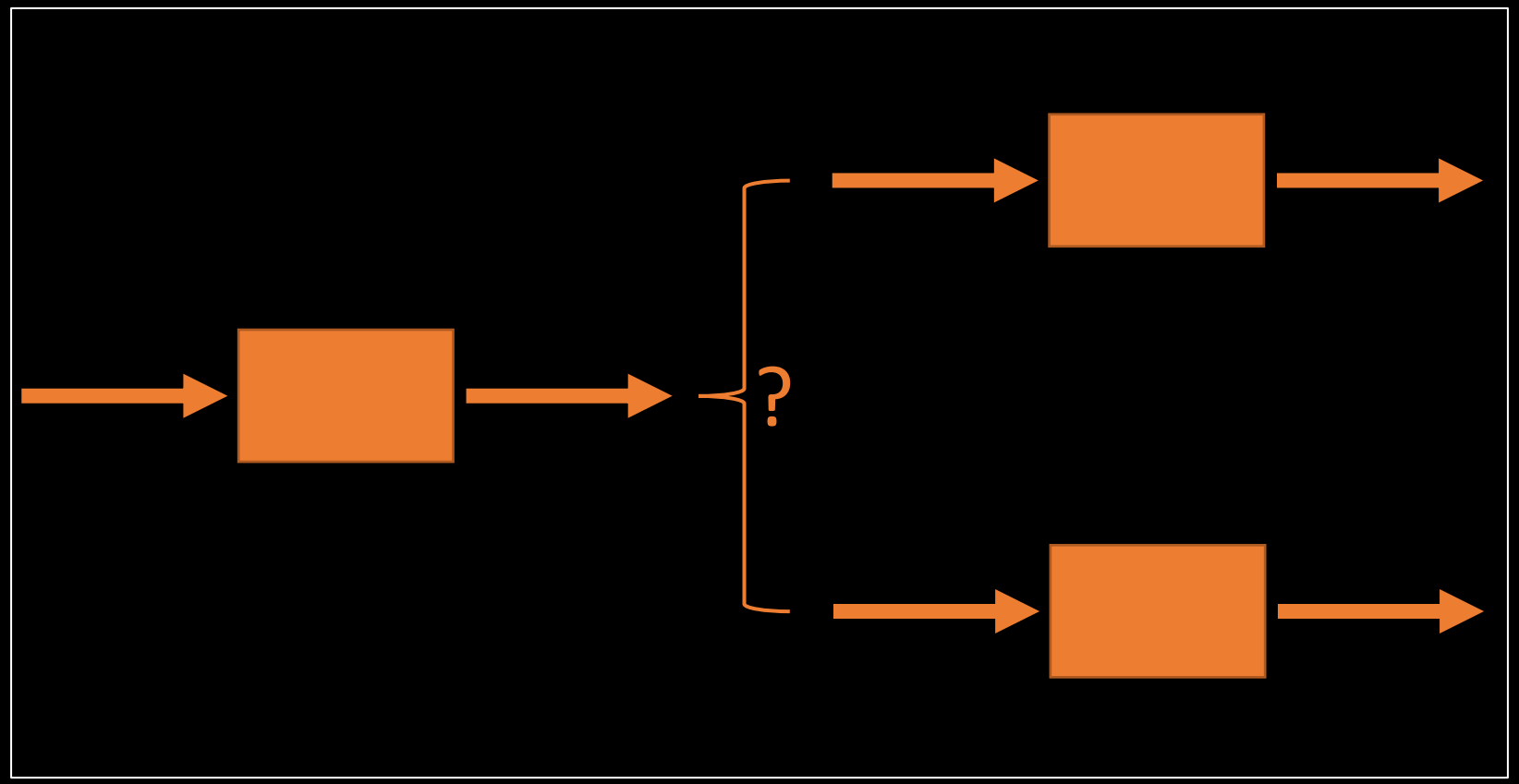

```
a = fonction(a, b, c, d)
```

L’appel de fonction




```
a = fonction(a, b, c, d)
```

Le test



Logique de Boole

a	b	AND	OR	XOR

Logique de Boole

a	b	AND	OR	XOR
0	0			
0	1			
1	0			
1	1			

Logique de Boole

a	b	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Le programme

```
#include <stdio.h>

int main() {

    printf("Hello, Universe!"); //see big

    return 0;
}
```

Le programme

```
#include <stdio.h>

int main() {

    printf("Hello, Universe!"); //see big

    return 0;
}
```

EXERCICE – EXERCICE – EXERCICE

Faire un hello world.

EXERCICE – EXERCICE – EXERCICE

La boucle

```
#include <stdio.h>

int main() {
    int i = 10;
    while(i>0) {
        printf("Hello, multiverse!");
        i = i-1;
    }
    return 0;
}
```

La boucle

```
#include <stdio.h>

int main() {
    int i = 10;
    while(i>0) {
        printf("Hello, multiverse!");
        i = i-1;
    }
    return 0;
}
```

EXERCICE – EXERCICE – EXERCICE

Afficher les nombres de 1 à 16 à l'aide d'une boucle.

EXERCICE – EXERCICE – EXERCICE

La fonction

```
#include <stdio.h>

void hellos(int numberOfHellos){
    int i=0;
    while(i<numberOfHellos) {
        printf("Hello, multiverse!");
        i++;;
    }
}


int main() {

    hellos(15);

    return 0;
}
```

La fonction

```
#include <stdio.h>

void hellos(int numberOfHellos){
    int i=0;
    while(i<numberOfHellos) {
        printf("Hello, multiverse!");
        i++;;
    }
}


int main() {

    hellos(15);

    return 0;
}
```

EXERCICE – EXERCICE – EXERCICE

Afficher les nombres de x à y à l'aide d'une fonction paramétrable.

EXERCICE – EXERCICE – EXERCICE

La boucle des fainéants

```
#include <stdio.h>

int main() {
    int i=10;
    for(i=10; i>0; i--) {
        printf("Hello, multiverse!");
    }
    return 0;
}
```

La boucle des fainéants

```c
#include <stdio.h>

int main() {
    int i=10;
    for(i=10; i>0; i--) {
        printf("Hello, multiverse!");
    }
    return 0;
}
```

EXERCICE – EXERCICE – EXERCICE

Afficher les nombres de x à y à l'aide d'une boucle de fainéant dans une fonction.

EXERCICE – EXERCICE – EXERCICE

Le test

```
#include <stdio.h>

int main() {
    int i = 10;
    while(i>0) {
        if(i%2==1){
            printf("Hello, odd multiverse!");
        }
        i = i-1;
    }
    return 0;
}
```

Le test

```
#include <stdio.h>

int main() {
    int i = 10;
    while(i>0) {
        if(i%2==1){
            printf("Hello, odd multiverse!");
        }
        else if(i%2==0){
            printf("Hello, even multiverse!"
        }
        i = i-1;
    }
    return 0;
}
```

Le test

```
#include <stdio.h>

int main() {
    int i = 10;
    while(i>0) {
        if(i%2==1){
            printf("Hello, odd multiverse!");
        }
        else if(i%2==0){
            printf("Hello, even multiverse!"
        }
        i = i-1;
    }
    return 0;
}
```

Le test

```
#include <stdio.h>

int main() {
    int i = 10;
    while(i>0) {
        if(i%2==1){
            printf("Hello, odd multiverse!");
        }
        else if(i%2==0){
            printf("Hello, even multiverse!"
        }
        else{
            printf("Ok, Houston we've had a problem here.");
            return 1;
        }
        i = i-1;
    }
    return 0;
}
```

Le test

EXERCICE – EXERCICE – EXERCICE

Afficher les tous les nombres premiers jusqu'à 500.

EXERCICE – EXERCICE – EXERCICE

```
#include <stdio.h>

int main( ) {
    int i = 10;
    while(i>0) {
        if(i%2==1){
            printf("Hello, odd
multiverse!");
        }
        else if(i%2==0){
            printf("Hello, even
multiverse!"
        }
        else{
            printf("Ok, Houston
we've had a problem here.");
            return 1;
        }
        i = i-1;
    }
    return 0;
}
```

Le switch

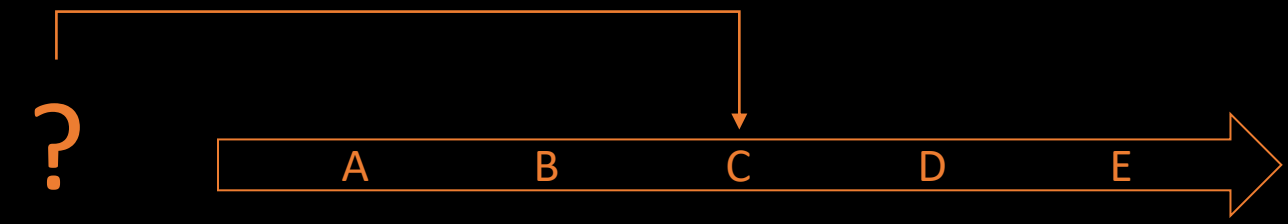
```
#include <stdio.h>
int main() {
    int i = 0;
    while(i<10) {
        switch(i){
            case 1:
                printf("one reached");
                break;
            case 2:
                printf("two reached");
                break;
            case 4:
                printf("four reached");
                break;
        }
        i++;
    }
    return 0;
}
```

Le ~~switch~~ goto

```c
#include <stdio.h>
int main() {
    int i = 0;
    while(i<10) {
        switch(i){
            case 1:
                printf("one reached");
                break;
            case 2:
                printf("two reached");
                break;
            case 4:
                printf("four reached");
                break;
        }
        i++;
    }
    return 0;
}
```

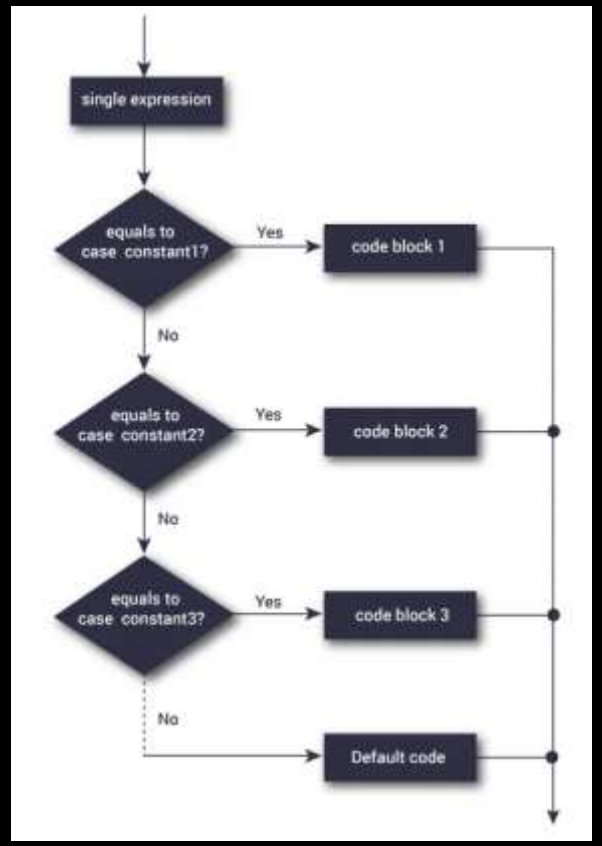
Le « test » goto

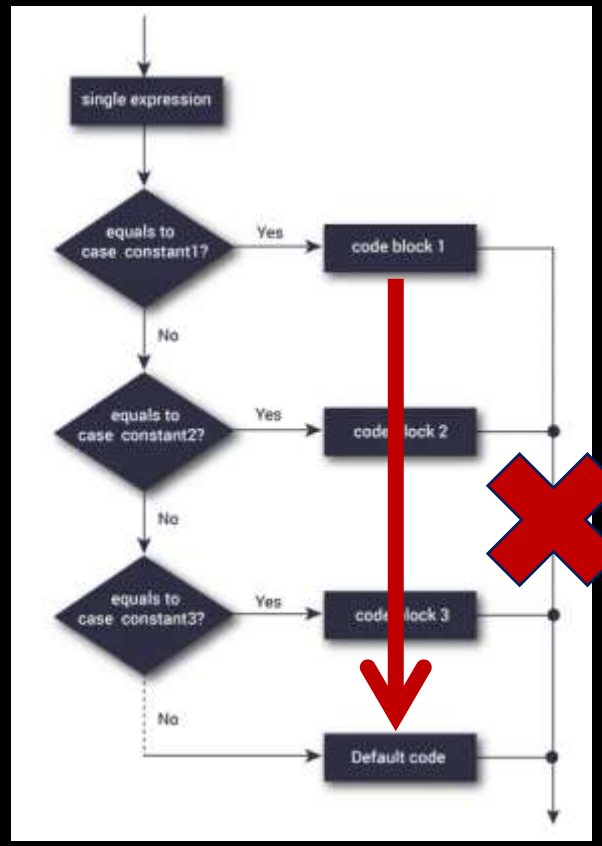
```c
#include <stdio.h>
int main() {
    int i = 0;
    while(i<10) {
        switch(i){
            case 1:
                printf("one reached");
                break;
            case 2:
                printf("two reached");
                break;
            case 4:
                printf("four reached");
                break;
        }
        i++;
    }
    return 0;
}
```



Le « test » goto

Google : “switch case c” → https://www.programiz.com/c-programming/c-switch-case-statement





Le « test » goto

```c
#include <stdio.h>
int main() {
    int i = 0;
    while(i<10) {
        switch(i){
            case 1:
                printf("one reached");
                break;
            case 2:
                printf("two reached");
                break;
            case 4:
                printf("four reached");
                break;
        }
        i++;
    }
    return 0;
}
```

```c
#include <stdio.h>
int main() {
    int i = 0;
    while(i<10) {
        switch(i){
            case 4:
                printf("four reached");
            case 2:
                printf("two reached");
            case 1:
                printf("one reached");
        }
        i++;
    }
    return 0;
}
```

Le « test » goto

```c
#include <stdio.h>
int main() {
    int i = 0;
    while(i<10) {
        switch(i){
            case 1:
                printf("one reached");
                break;
            case 2:
                printf("two reached");
                break;
            case 4:
                printf("four reached");
                break;
        }
        i++;
    }
    return 0;
}
```

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

EXERCICE – EXERCICE – EXERCICE

Utiliser un « test goto » pour afficher :
Si i=1 : « hello universe »
Si i=2 : « hello universes »
Si i=3 : « hello multiverse »
Si i>4 : les nombres premiers de i à 100×i

EXERCICE – EXERCICE – EXERCICE

```c
    return 0;
}
```

L’instruction pré-processeur

```c
#include <stdio.h>
#define NUMBERHELLOS 10
void hellos(int numberOfHellos){
    int i=0;
    while(i< NUMBERHELLOS) {
        printf("Hello, multiverse!");
        i++;;
    }
}


int main() {

    hellos(15);

    return 0;
}
```

L’instruction pré-processeur

main.c

```c
#include <stdio.h>
#include "hellogen.h"

int main( ) {

    hellos(15);

    return 0;
}
```

hellogen.h

```c
#ifndef HELLOGEN_H
#define HELLOGEN_H

void hellos(int numberOfHellos);
#endif /*HELLOGEN_H*/
```

hellogen.c

```c
#include "hellogen.h“
#define NUMBERHELLOS 10

void hellos(int numberOfHellos){
    int i=0;
    while(i< NUMBERHELLOS) {
        printf("Hello, multiverse!");
        i++;;
    }
}
```