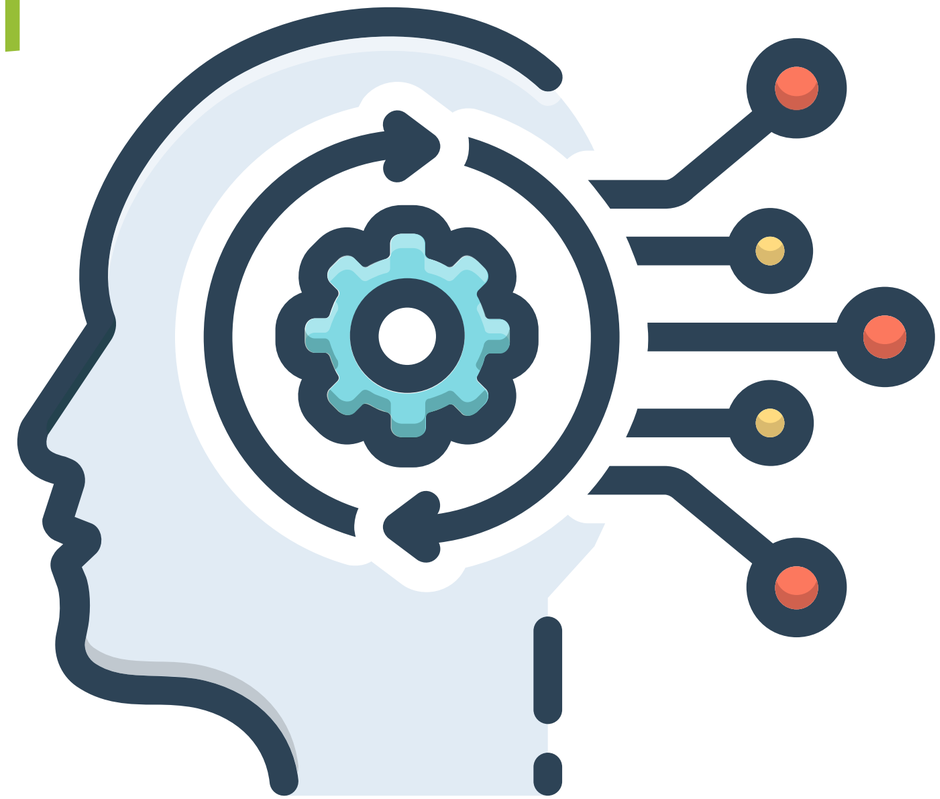
USSEIN

DR B. DIOP
babacar.diop@ussein.edu.sn

Algorithmes et Programmation II

Fonctions



INFO – AGROTIC

Last update: oct. 2025

Objectifs globaux du cours

A l'issue de de ce cours, vous devez être en mesure de :

- Comprendre la notion de fonction
- Classer les fonctions selon valeur de retour et arguments
- Savoir écrire et utiliser les 4 cas fondamentaux

Qu’est ce qu’une fonction en programmation ?

En programmation, **une fonction est un bloc de code nommé qui effectue une tâche spécifique**. Elle peut recevoir des données en entrée (appelées paramètres), les traiter, et retourner un résultat.

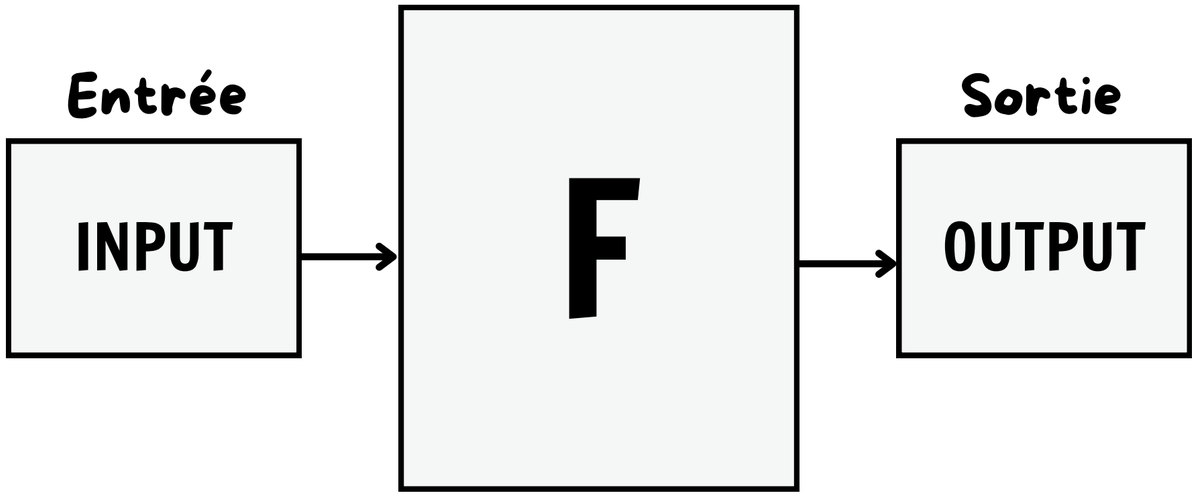


Schéma illustratif d’une fonction avec entrée et sortie

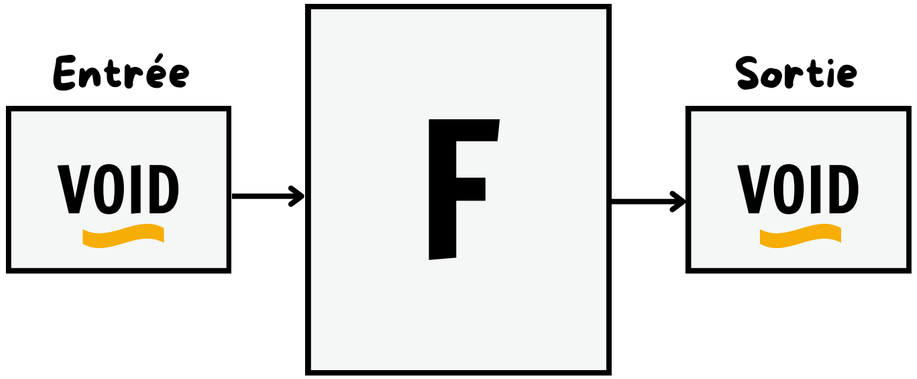
Les fonctions servent à organiser le code, à le rendre réutilisable et plus lisible, en évitant la répétition des mêmes instructions.

Pourquoi des fonctions ?

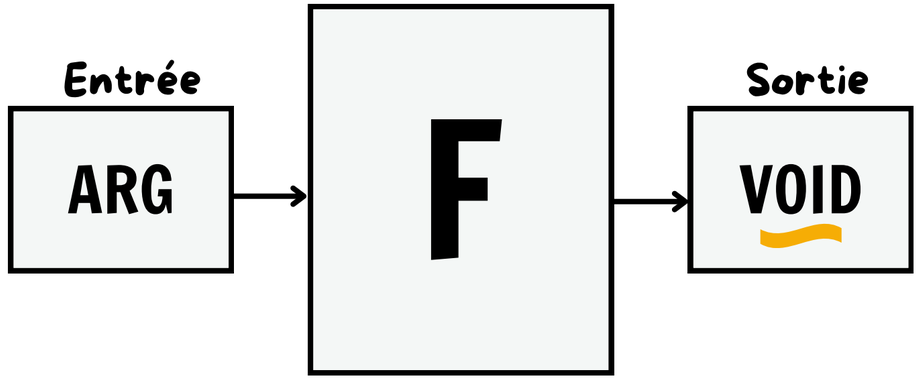
1. Réutilisabilité du code
2. Lisibilité et modularité
3. Test unitaire plus facile
4. Encapsulation d'une logique / abstraction

Catégories de fonctions usuelles

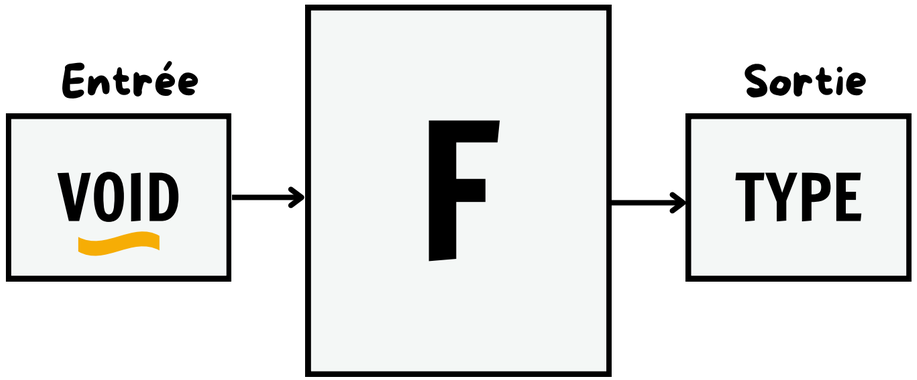
FONCTION sans ARGUMENT et sans RETOUR



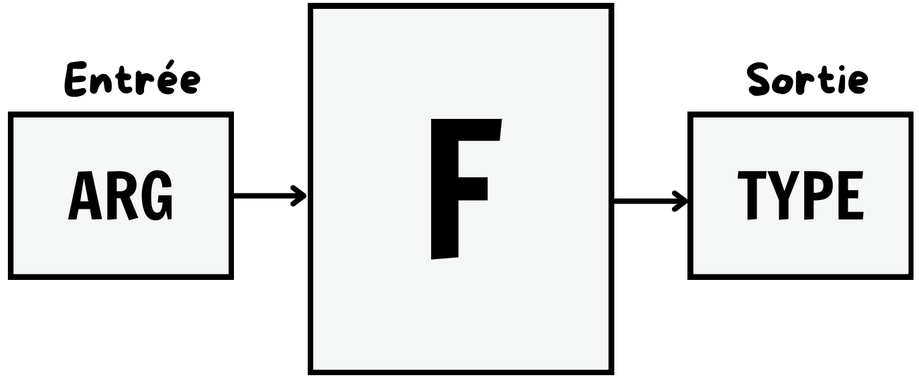
FONCTION avec 1 ARGUMENT et sans RETOUR



FONCTION sans ARGUMENT avec RETOUR

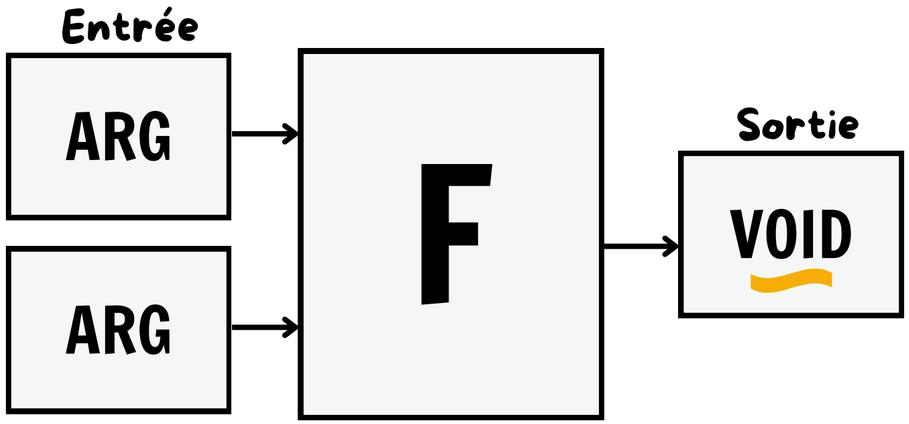


FONCTION avec 1 ARGUMENT avec RETOUR

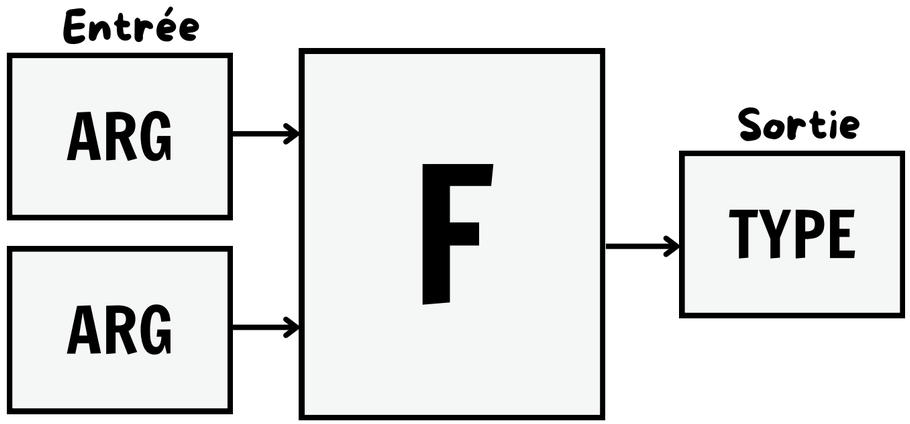


Cas de fonctions avec 2 arguments

FONCTION avec 2 ARGUMENTS et sans RETOUR

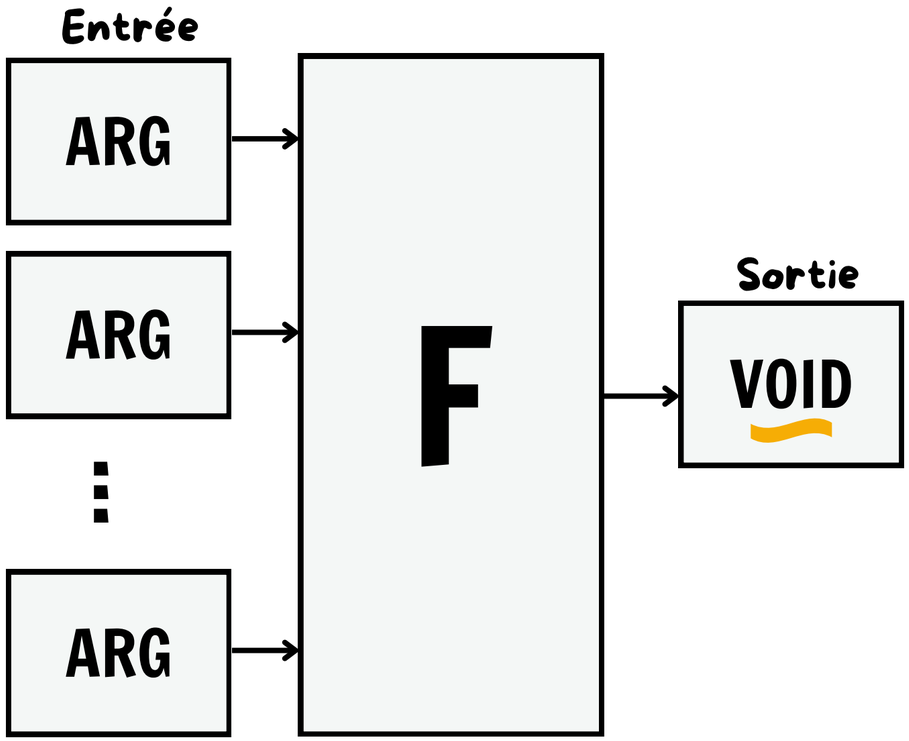


FONCTION avec 2 ARGUMENTS avec RETOUR

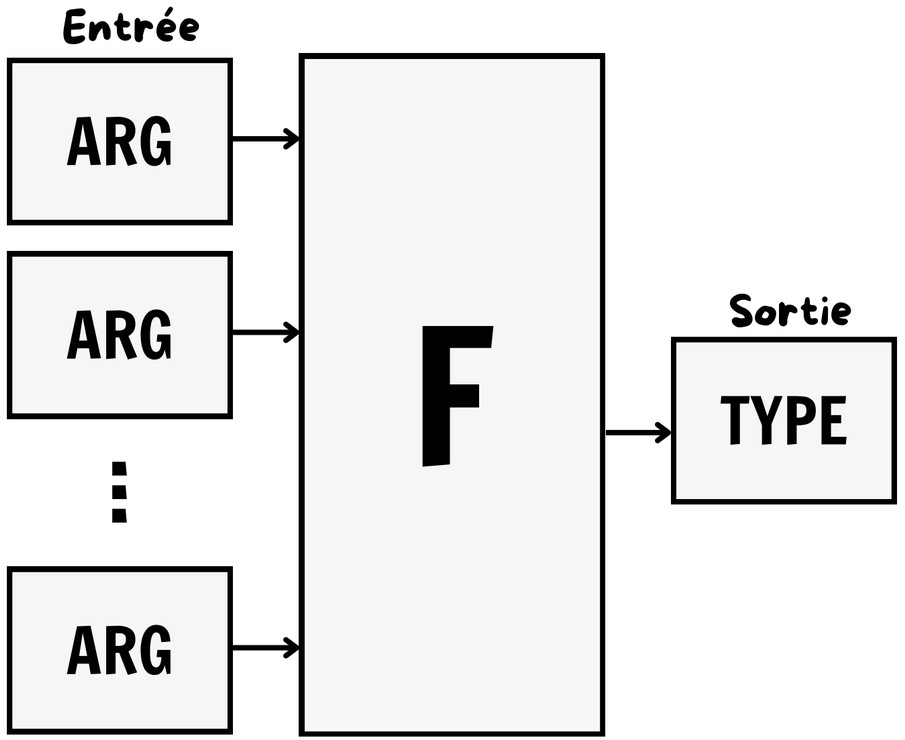


Cas de fonctions avec N arguments

FONCTION avec N ARGUMENTS et sans RETOUR



FONCTION avec N ARGUMENTS avec RETOUR



Structure d'une fonction en C

Les éléments constituant une fonction sont :

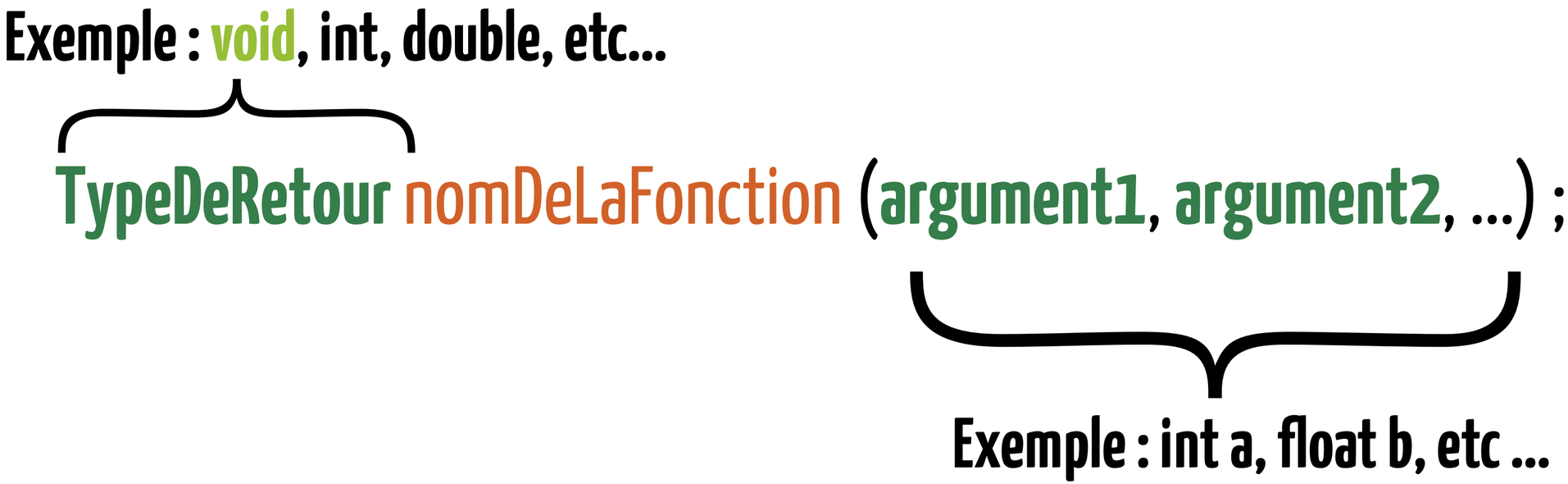
- Un type de retour (ex : void, int, double)
- Un nom pour désigner la fonction et qui sera utilisé à l'appel dans main()
- Une liste d'arguments entre parenthèses séparés par des virgules (ou vide)
- Une suite d'instructions entre accolades comme dans un code normal

```
TypeDeRetour nomDeLaFonction (argument1, argument2, ...) {
        Corps de la fonction
}
```

Signature d'une fonction en C

Les éléments constituant une fonction sont :

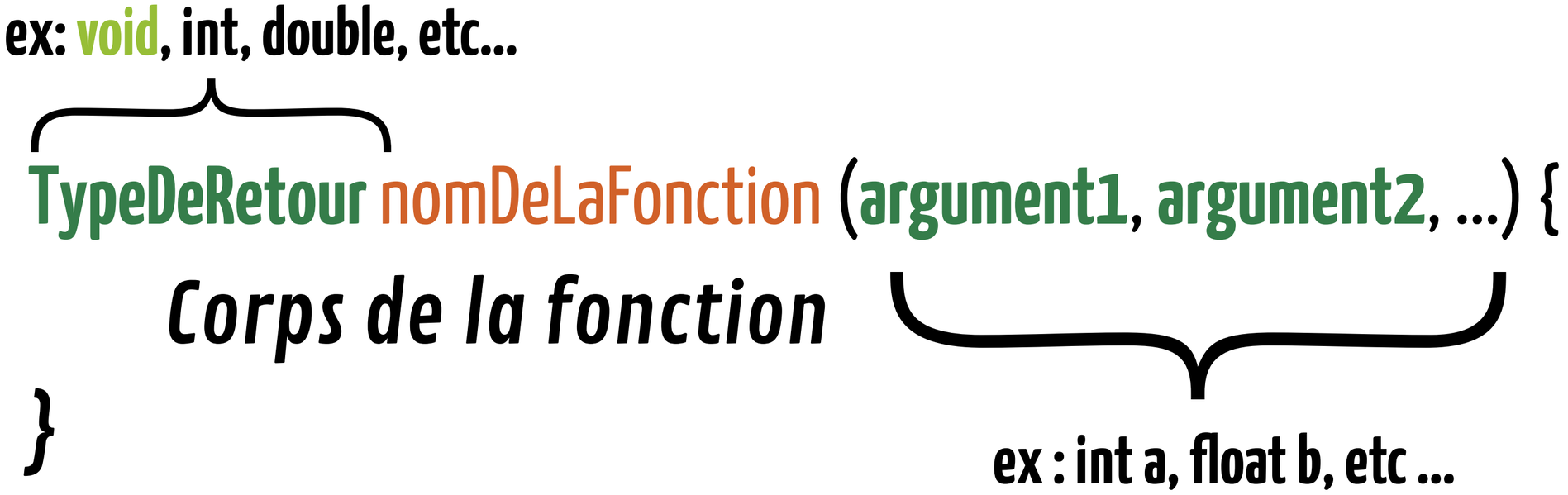
- Un type de retour (ex : void, int, double)
- Un nom pour désigner la fonction et qui sera utilisé à l'appel dans main()
- Une liste d'arguments entre parenthèses séparés par des virgules (ou vide)



Structure générale d'une fonction en C

Les éléments constituant une fonction sont :

- Un type de retour (ex : void, int, double)
- Un nom pour désigner la fonction et qui sera utilisé à l'appel dans main()
- Une liste d'arguments entre parenthèses séparés par des virgules (ou vide)
- Une suite d'instructions entre accolades comme dans un code normal



Classification des fonctions

On peut classer les fonctions par paire (**valeur de retour**, **nombre d'arguments**) en 4 catégories intuitives :

- **(0,0)** : **void** + **aucun argument** — fonction sans paramètre et sans valeur de retour
- **(0,1)** : **void** + **arguments** — fonction paramétrée et sans valeur de retour
- **(1,0)** : **valeur de retour** + **aucun argument** — fonction sans paramètre et avec retour
- **(1,1)** : **valeur de retour** + **arguments** — fonction paramétrée et avec valeur de retour

Cas (0,0) – Fonction sans argument et sans valeur de retour

Une fonction qui ne prend pas de paramètre et qui ne retourne rien effectue souvent une suite d'action qui n'a pas besoin d'information venant de l'utilisateur.

```c
#include <stdio.h>

void dire_bonjour(void) {
    printf("Bonjour !\n");
}

int main(void) {
    dire_bonjour();
    return 0;
}
```

Cas (0,0) – Fonction sans argument et sans valeur de retour

Comprendre

void dans la liste d'arguments et void comme valeur de retour

```c
#include <stdio.h>

void dire_bonjour(void) {
    printf("Bonjour !\n");
}

int main(void) {
    dire_bonjour();
    return 0;
}
```

```c
void dire_bonjour(void);
```

void dans la liste d'arguments indique explicitement qu'aucun paramètre n'est attendu.

Cas (0,0) – Fonction sans argument et sans valeur de retour

Comprendre

void dans la liste d'arguments et void comme valeur de retour

```c
#include <stdio.h>

void dire_bonjour(void) {
    printf("Bonjour !\n");
}

int main(void) {
    dire_bonjour();
    return 0;
}
```

Cas (0,0) – Fonction sans argument et sans valeur de retour

Appel de la fonction

```c
#include <stdio.h>

void dire_bonjour(void) {
    printf("Bonjour !\n");
}

int main(void) {
    dire_bonjour();
    return 0;
}
```

dire_bonjour();

Une fonction sans paramètre, ni valeur de retour s'appelle en évoquant le **nom de la fonction** suivie de **parenthèses vides**.

Cas (0,1) – Fonction avec argument et sans valeur de retour

Une fonction qui prend des arguments et ne retourne pas de valeur réalise souvent une action (opération) en utilisant des paramètres et affiche le résultat à l'écran.

```c
#include <stdio.h>

void afficher_somme(int a, int b) {
    printf("%d + %d = %d\n", a, b, a + b);
}

int main(void) {
    afficher_somme(3, 7);
    return 0;
}
```

Par exemple, dans cette fonction, on demande de **calculer la somme** de deux entiers et d'**afficher le résultat**.

Cas (0,1) – Fonction avec argument et sans valeur de retour

Comprendre

la liste des arguments et void comme valeur de retour

```c
#include <stdio.h>

void afficher_somme(int a, int b) {
    printf("%d + %d = %d\n", a, b, a + b);
}

int main(void) {
    afficher_somme(3, 7);
    return 0;
}
```

```c
void somme(int a , int b) {
```

Indique que la fonction prend deux paramètres. Ces derniers sont toujours séparés par des virgules (,).

Cas (0,1) – Fonction avec argument et sans valeur de retour

Comprendre

la liste des arguments et **void** comme valeur de retour

```c
#include <stdio.h>

void afficher_somme(int a, int b) {
    printf("%d + %d = %d\n", a, b, a + b);
}

int main(void) {
    afficher_somme(3, 7);
    return 0;
}
```

```c
void somme(int a , int b);
```

Indique que la fonction ne va pas avoir de valeur de retour. Le résultat sera affiché.

Cas (0,1) – Fonction avec argument et sans valeur de retour

Comment appelle-t-on une telle fonction dans main() ? Méthode 1

```c
#include <stdio.h>

void afficher_somme(int a, int b) {
    printf("%d + %d = %d\n", a, b, a + b);
}

int main(void) {
    afficher_somme(3, 7);
    return 0;
}
```

Pour appeler une telle fonction :

- En évoquant directement la fonction par son nom et en listant les paramètres sous forme de constantes.

Cas (0,1) – Fonction avec argument et sans valeur de retour

Comment appelle-t-on une telle fonction dans main() ? Méthode 2

```c
#include <stdio.h>

void afficher_somme(int a, int b) {
    printf("%d + %d = %d\n", a, b, a + b);
}

int main(void) {
    int x = 3;
    int y = 7;

    afficher_somme(x, y);

    return 0;
}
```

Pour appeler une telle fonction :

- On crée deux variables locales x et y dans main().
- On les transmet à la fonction afficher_somme().
- La fonction reçoit des copies de x et y et exécute son bout de code.

Cas (0,1) – Fonction avec argument et sans valeur de retour

Comprendre le passage par valeur

Passage par valeur en C : effets sur les variables originales

- En langage C, tous les arguments sont passés par valeur.
- Cela signifie que la fonction reçoit une copie des variables passées en paramètre.
- Toute modification à l'intérieur de la fonction n'affecte pas les variables originales.

Cas (0,1) – Fonction avec argument et sans valeur de retour

Passage par valeur = copie de variable

```c
void incrementer(int n) {
    n = n + 1;
    printf("Dans la fonction : n = %d\\n", n);
}

int main(void) {
    int x = 5;
    incrementer(x);
    printf("Dans main : x = %d\\n", x);
    return 0;
}
```

L'exécution de ce code va donner la sortie ci-dessous :

```
Dans la fonction : n = 6
Dans main : x = 5
```

On remarque que la valeur de **x** dans main n'a pas changé après l'appel de la fonction!

Cas (0,1) – Fonction avec argument et sans valeur de retour

Passage par adresse (pointeur) = adresse mémoire de la variable

```c
void incrementer(int *p) {
    (*p)++;
}

int main(void) {
    int x = 5;
    incrementer(&x);
    printf("x = %d\\n", x); // x = 6
}
```

Si on veut modifier la valeur originale, on doit passer l'adresse de la variable (passage par pointeur)

Remarquer le pointeur en argument

Ajouter le symbole &

Cas (1,0) – Fonction sans argument et valeur de retour

Une fonction qui ne reçoit aucun argument et renvoie une valeur effectue souvent une suite d'opérations dont le résultat sera renvoyé au programme qui l'appelle.

```c
#include <stdlib.h>
#include <time.h>

int tirer_aleatoire(void) {
    return rand() % 100; // 0..99
}

int main(void) {
    srand((unsigned) time(NULL));
    int x = tirer_aleatoire();
    return 0;
}
```

Discussion : effets de l'état externe (rand, variables globales).

Cas (1,0) – Fonction sans argument et valeur de retour

Exemple de fonction

Générer un nombre entier et retourner ce nombre en utilisant la fonction rand() en C.

```c
#include <stdlib.h>
#include <time.h>

int tirer_aleatoire(void) {
    return rand() % 100; // 0..99
}

int main(void) {
    srand((unsigned) time(NULL));
    int x = tirer_aleatoire();
    return 0;
}
```

```c
int tirer_aleatoire(void) {
```

Indique que la fonction ne prend aucun argument.

Cas (1,0) – Fonction sans argument et valeur de retour

Exemple de fonction

Générer un nombre entier et retourner ce nombre en utilisant la fonction rand() en C.

```c
#include <stdlib.h>
#include <time.h>

int tirer_aleatoire(void) {
    return rand() % 100; // 0..99
}

int main(void) {
    srand((unsigned) time(NULL));
    int x = tirer_aleatoire();
    return 0;
}
```

```c
int tirer_aleatoire(void) {
```

Indique que la fonction va avoir comme valeur de retour un entier.

Cas (1,1) – Fonction avec arguments et valeur de retour

Une fonction qui prend en arguments des paramètres et qui retourne une valeur est une fonction classique qui transforme des entrées en sortie.

```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```

Discussion : pureté, absence d'effets secondaires, testabilité.

Cas (1,1) – Fonction avec arguments et valeur de retour

Exemple de fonction

Calculer le maximum entre deux nombres entiers et retourner le résultat

```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```

```c
int max(int a, int b) {
```

Indique que la fonction prend deux paramètres. Ces derniers sont toujours séparés par des virgules (,).

Cas (1,1) – Fonction avec arguments et valeur de retour

Exemple de fonction

Calculer le maximum entre deux nombres entiers et retourner le résultat.

```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```

```c
int max(int a, int b) {
```

Indique que la fonction va avoir comme valeur de retour un entier.

Cas (1,1) – Fonction avec arguments et valeur de retour

Exemple de fonction

Calculer le maximum entre deux nombres entiers et retourner le résultat.

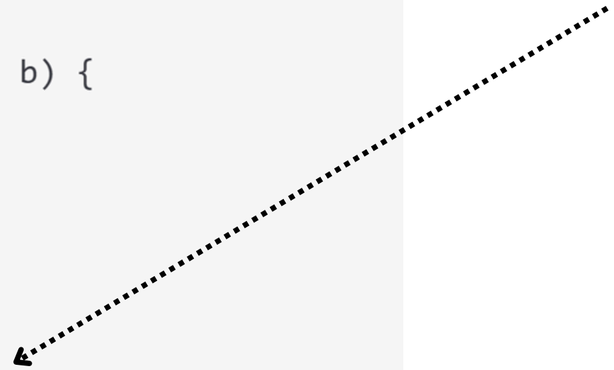
```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```



1.Préparation des arguments

Dans main(), on définit les variables à transmettre

Cas (1,1) – Fonction avec arguments et valeur de retour

Exemple de fonction

Calculer le maximum entre deux nombres entiers et retourner le résultat.

```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```

1.Préparation des arguments

Dans main(), on définit les variables à transmettre

2.Variable de récupération de résultat

Dans main(), on définit une variable de même type que le type de retour de la fonction

Cas (1,1) – Fonction avec arguments et valeur de retour

Exemple de fonction

Calculer le maximum entre deux nombres entiers et retourner le résultat.

```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```

1.Préparation des arguments

Dans main(), on définit les variables à transmettre

2.Variable de récupération de résultat

Dans main(), on définit une variable de même type que le type de retour de la fonction

3.Appel de la fonction

On appelle la fonction avec ces arguments. La fonction est exécutée arrivée à cette ligne, et le résultat stocké dans la variable **resultat**

Cas (1,1) – Fonction avec arguments et valeur de retour

Exemple de fonction

Calculer le maximum entre deux nombres entiers et retourner le résultat.

```c
#include <stdio.h>

int addition(int a, int b) {
    return a + b;
}

int main(void) {
    int x = 3, y = 7;
    int resultat;

    resultat = addition(x, y);
    printf("La somme est : %d\n", resultat);

    return 0;
}
```

4.Utilisation du résultat

Enfin, le programme peut afficher ou utiliser ce résultat autrement. Ici on affiche seulement !

Exercices d’application

1. Écrire une procédure (0,0) qui affiche un menu.
2. Écrire (0,1) : afficher une table de multiplication pour n donné.
3. Écrire (1,0) : fonction qui renvoie le timestamp actuel (secondes).
4. Écrire (1,1) : fonction qui prend un tableau et sa taille et retourne la moyenne (double).
5. Bonus : écrire une fonction qui retourne deux valeurs (utiliser pointeurs).
6. Fonction récursive pour calculer la factorielle.
7. Fonction pour inverser une chaîne de caractères.
8. Procédure qui trie un tableau par sélection.
9. Fonction qui teste si un nombre est premier.