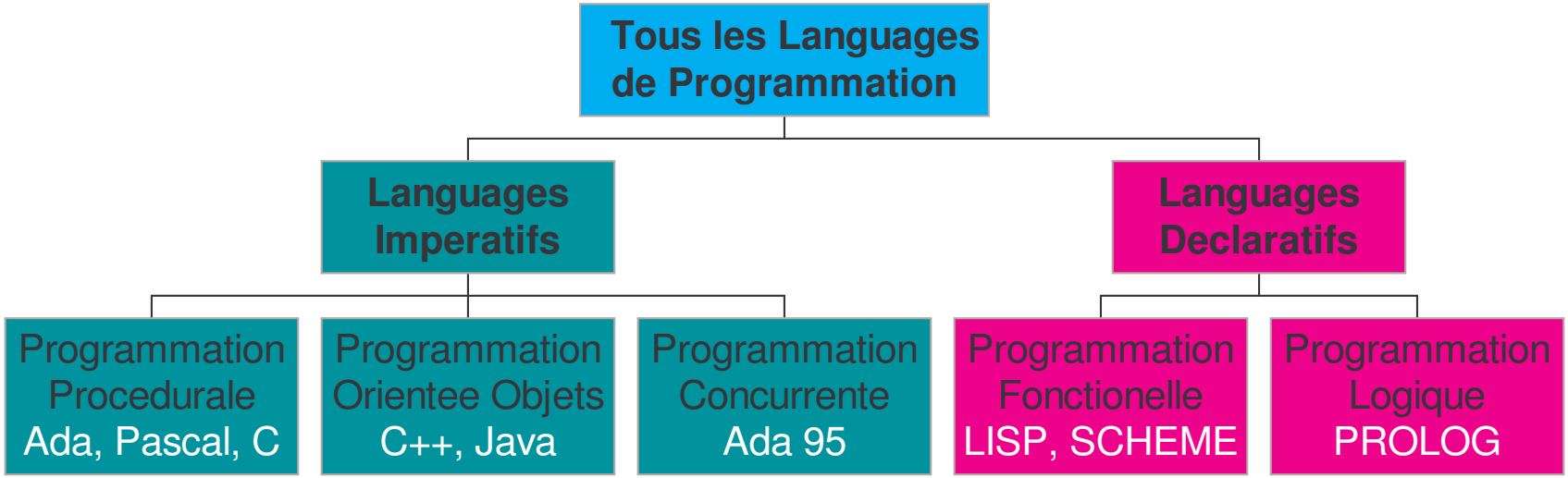
la programmation en langage C

Classification



Languages Imperatifs: Langages incluant des moyens pour le programmeur d 'attribuer des valeurs a des locations en mémoire.
Languages Declaratifs: Langages pour lesquels le programmeur réfléchit en terme de valeurs des fonctions et de relations entre entités diverses. Il n 'y a pas d 'attribution de valeurs aux variables.

Caractèristique du C

- Structuré
- Modulaire: peut être découpé en modules qui peuvent être compilés séparement
- Universel: n'est pas orienté vers un domaine d'application particulier
- Typé: tout objet C doit être déclaré avant d’être utilisé
- Portable: sur n'importe quel système en possession d'un compilateur C

Un long fleuve tranquille

C
fichier

Compilateur C

Code
assembleur

Assembleur

Code
objet

Linker

Code
executable

Fichier C (extension .c)

```
/* exemple de programme C :
- somme des nb de 1 à 10 et affichage
de la valeur*/
 #include <stdio.h>
 int main (void)
 {
    int somme; int i;
    somme = 0;
    for (i = 1;  i <= 10; i++)
{
        somme = somme + i;
     }
   printf ("%d\n", somme);
    somme = 0;
}
```

0

1

2

3

4

5

En C le programme principal s'appelle toujours *main*

déclarations de variables de type entier (cases mémoire pouvant contenir un entier)

instruction d'affectation de valeur à la variable *somme*

instructions exécutées en séquence

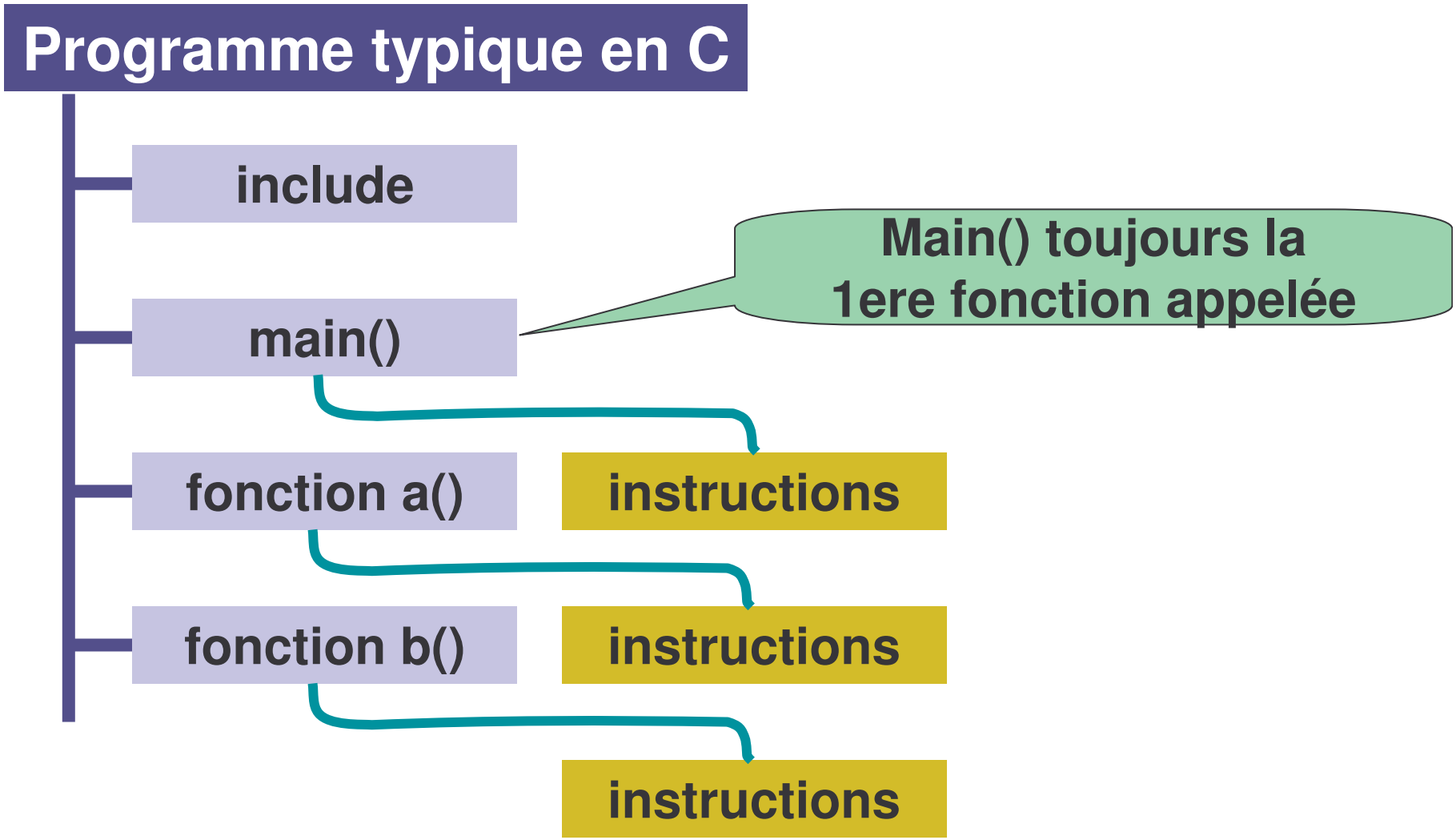
l'instruction entre accolades est exécutée pour les valeurs de *i* allant de 1 à 10

affiche à l'écran la valeur de l'entier contenu dans *somme*

Ecrire le programme suivant :

```
#include <stdio.h>
int main(void) {
    int a= 257, b = 381;
    if (a > 0 && b > 0) {
        printf("  PGCD(%3d,%"d)\n",a,b);
        while (a != b) {
            if (a < b) b = b-a;
            else a = a-b;
            printf("=PGCD(%3d,%3d)\n",a,b);
        }
        printf("=%d\n",a);
    }
    return 0;
}
```

Anatomie


Programme typique en C
include
main()
Main() toujours la
1ere fonction appelée
fonction a()
instructions
fonction b()
instructions
instructions

Commandes simples de compilation

- Prétraitement, compilation et édition de liens :
- *gcc –Wall –g pgcd.o –lm –o pgcd*
- l'option *–Wall* demande une compilation avec des diagnostics sur la propreté du code
- l'option *–g* demande que la *table des symboles* soit ajoutée à l'exécutable
- l'option *–lm* demande de lier la librairie mathématique
- l'option *–o pgcd* demande que le résultat (l'exécutable) soit nommé pgcd au lieu de a.out
- Le programme est à lancer avec ./pgcd

Types d'instruction en C

- Déclarations des variables
- Assignations
- Fonctions
- Contrôle

Types de données et de variables

- Déclaration des variables
  - int y;
  - char yesno, ok;
  - int ordered = 1, onhand = 0;
  - float total = 43.132;
  - char *cptr = NULL;

Types de données et de variables

- Type « char » ou « signed char »:
  - ASCII sur 32 bits
    - de -2147483648 à 2147483647.

- Type « unsigned char »:
  - ASCII sur 32 bits
    - de 0 à 4294967295.

Types de données et de variables

- Type « short » ou « signed short »
  - en complément à 2 sur 32 bits
    - de -2147483648 à 2147483647.

- Type « unsigned short »:
  - binaire sur 32 bits
    - de 0 à 4294967295.

Types de données et de variables

- Type « int » ou « signed int »
  - en complément à 2 sur 32 bits
    - de -2147483648 à 2147483647.

- Type « unsigned int »:
  - binaire sur 32 bits
    - de 0 à 4294967295.

Types de données et de variables

- Type « long » ou « signed long »
  - en complément à 2 sur 32 bits
    - de -2147483648 à 2147483647.

- Type « unsigned long »:
  - binaire sur 32 bits
    - de 0 à 4294967295.

Types de données et de variables

- Type « enum » (ordered list)
  - en complément à 2 sur 32 bits
    - de -2147483648 à 2147483647.

- Type « float »
  - format TMS320C30 sur 32 bits
    - de 5.9×10^{-39} à 3.4×10^{38}.

Types de données et de variables

- Type « double »
  - format TMS320C30 sur 32 bits
    - de 5.9×10^{-39} à 3.4×10^{38}.

- Type « long double »
  - format TMS320C30 sur 40 bits
    - de 5.9×10^{-39} à 3.4×10^{38}.

Types de données et de variables

- Type « pointer »
  - binaire sur 32 bits
    - de 0 à 0xFFFFFFFF.

Les pointeurs

- Un pointeur contient l'adresse d'une autre variable.

- Déclaration:
  - float *wirelen;

Les pointeurs

- Utilisation:
  - wirelen = &wire2 ;
    - Contenu de wirelen = adresse de wire2
  - *wirelen = 30.5 ;
    - Même effet que wire2 = 30.5.

Les pointeurs et les vecteurs

- Déclaration:
  - float *arrayptr;
  - float farray[30];

- Utilisation:
  - arrayptr = farray; ou arrayptr = &farray[0];

Les pointeurs et les vecteurs

- Accès à une valeur dans le vecteur:
  - *(arrayptr+3) équivalent à farray[3]

- Balayage simple avec ++arrayptr

Les modificateurs des classes de mémorisation

- « extern »:
  - indique une variable ou une fonction déclarée dans un autre module.

- « register »:
  - demande au compilateur de placer des variables dans les registres du CPU. Augmente la vitesse de traitement.

Les modificateurs des classes de mémorisation

- « const »:
  - indique que le contenu d'une variable ne doit pas être modifiée.

- « volatile »:
  - indique qu'une variable peut voir son contenu changer à tout moment par le programme, des interruptions ou tout autre facteur extérieur. Empêche le compilateur de faire des optimisations sur cette variable.

Opérateurs et expressions

- Opérateurs à un paramètre:
  - - change le signe de la variable
  - ~complément à 1
  - ** « indirection » (pointeurs)*
    - *value = *salary; /* contenu pointé par salaire */*
  - &adresse
  - ++/-- incrémentation/décrémentation
  - sizeof()

Opérateurs et expressions

- Opérateurs arithmétique:
  - *,/,+,-
  - % modulo
- Opérateurs sur bits:
  - <<,>> décalage à gauche ou à droite
    - status = byte << 4;
  - &et
  - | ou
  - ^ ou exclusif

Opérateurs et expressions

- Opérateurs relationnels:
  - <,>,<=,=>
- Opérateurs d'égalité:
  - ==, !=
- Opérateurs logiques:
  - && et
  - || ou

Opérateurs et expressions

- Opérateur conditionnel:
  - result = mode > 0 ? 1 : 0;
  - if mode>0 then result=1 else result=0.

- Opérateurs d'assignation:
  - =, *=, /=, %=, +=, -=, <<=, >>=, &=, |=, ^=

Fonctions en C

- Plusieurs fonctions pré-définies:
  - printf(), sin(), atoi(), …
- Le prototype de ces fonctions sont dans fichiers d'entête (header file)
  - printf() dans stdio.h
  - sin() dans math.h

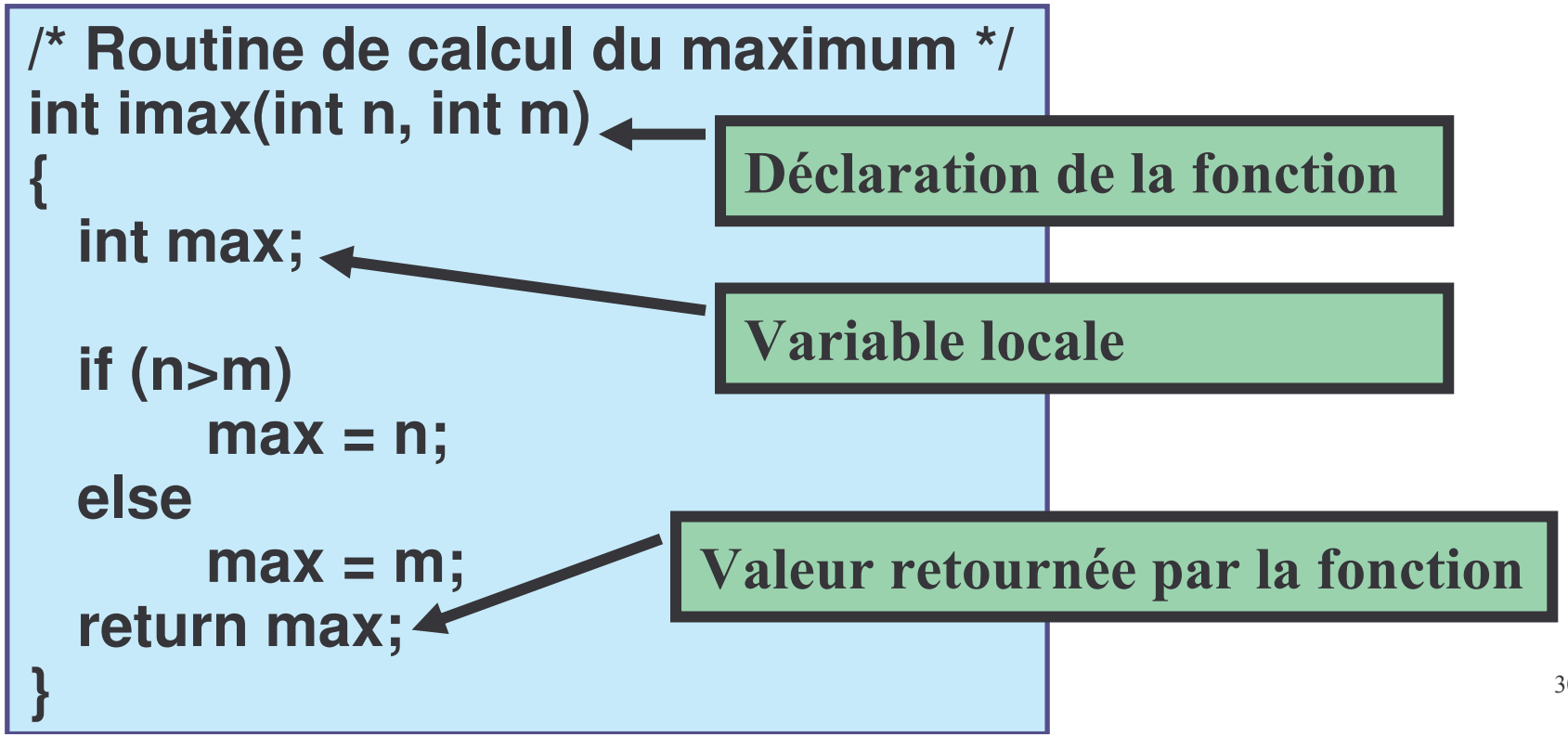
Fonctions en C

- Extrait de stdio.h

```
/*****************************************************************/
/* FORMATTED INPUT/OUTPUT FUNCTIONS                              */
/*****************************************************************/
extern int    fprintf(FILE *_fp, const char *_format, ...);
extern int    fscanf(FILE *_fp, const char *_fmt, ...);
extern int    printf(const char *_format, ...);
extern int    scanf(const char *_fmt, ...);
extern int    sprintf(char *_string, const char *_format, ...);
extern int    sscanf(const char *_str, const char *_fmt, ...);
extern int    vfprintf(FILE *_fp, const char *_format, char *_ap);
extern int    vprintf(const char *_format, char *_ap);
extern int    vsprintf(char *_string, const char *_format, char *_ap);
```

Fonctions en C

- Bien sûr, nous pouvons écrire nos propres fonctions.



Fonctions en C

- Fonctions sans arguments et ne retournant pas de valeur.
  - void fonction(void)

- Fonctions avec arguments ne retournant pas de valeur.
  - void fonction(int x, int y, char ch)

Fonctions en C

- Les fonctions exigent la déclaration d'un prototype avant son utilisation:

```
/* Programme principal */
#include <stdio.h>
int imax(int,int);
main()
{   …   }

int imax(int n, int m)
{   …   }
```

Prototype de la fonction

La fonction est définie ici

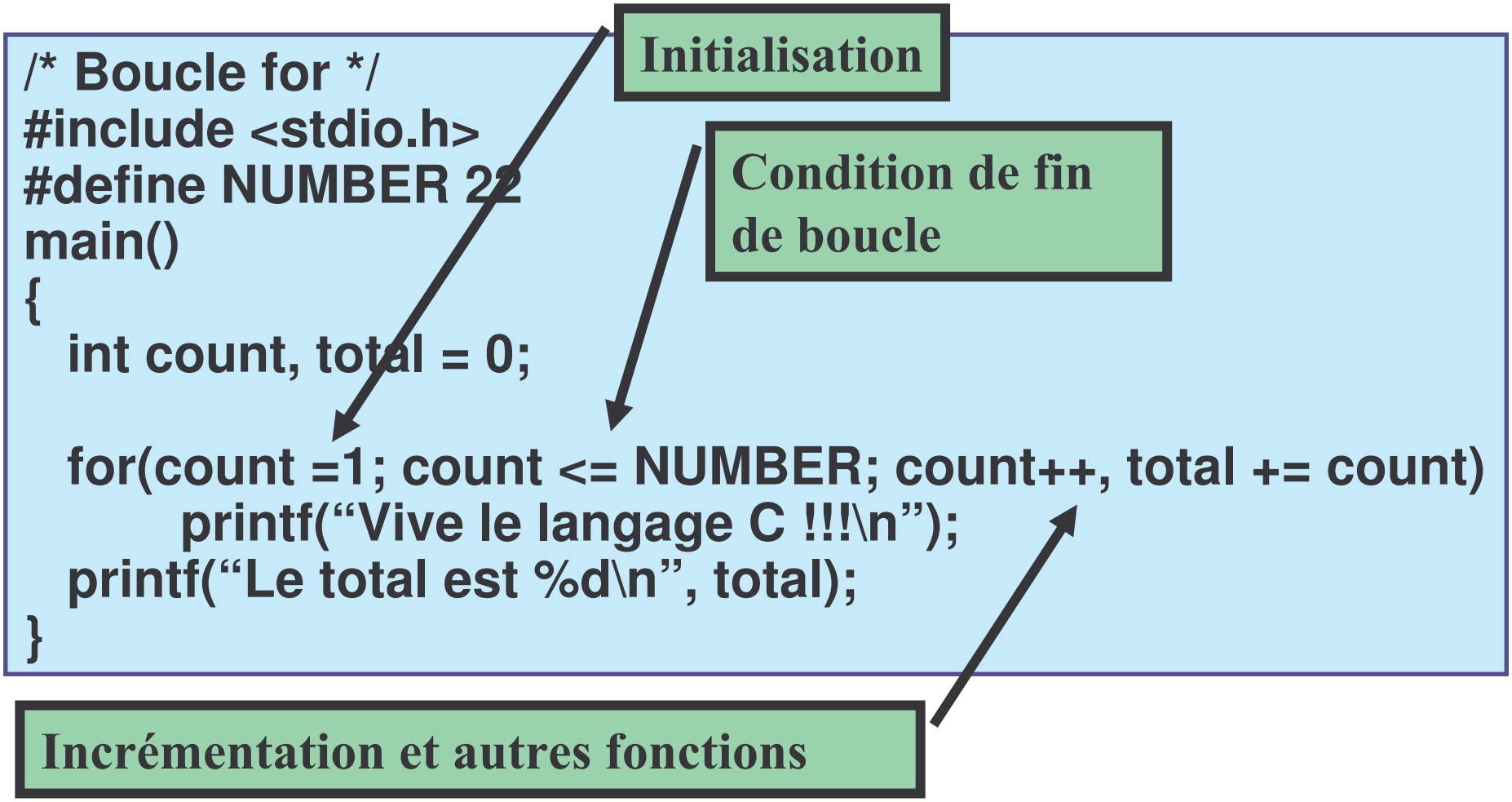
Fonctions en C

- La récursivité

```
/* Programme principal */
#include <stdio.h>
void up_and_down(int);
main()
{
  up_and_down(1)
 }

void up_and_down(int n)
{
  printf("Niveau %d\n" n);
  if (n<4)
      up_and_down(n+1);
  printf("NIVEAU %d\n" n);
 }
```

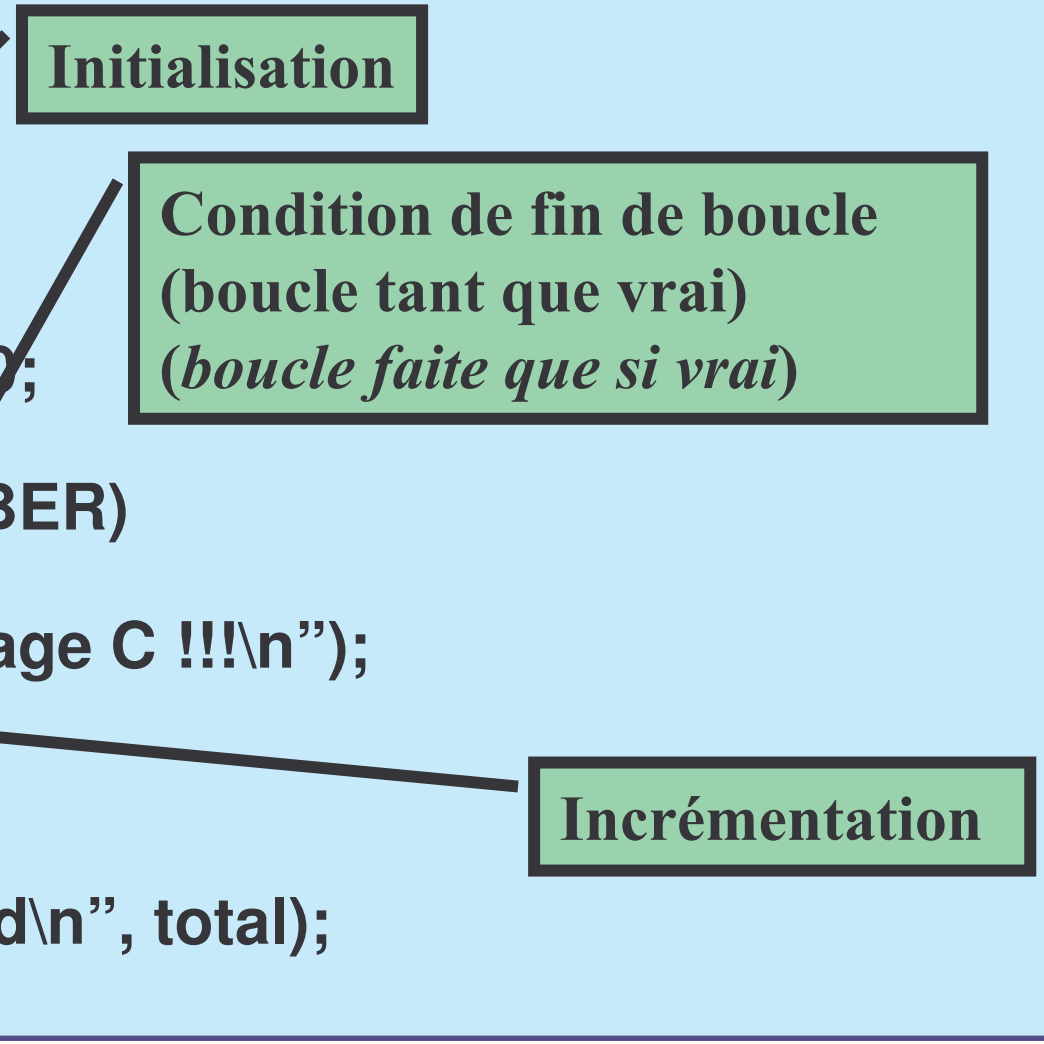
```
Niveau 1
Niveau 2
Niveau 3
Niveau 4
NIVEAU 4
NIVEAU 3
NIVEAU 2
NIVEAU 1
```

Boucle « for »


/* Boucle for */
#include <stdio.h>
#define NUMBER 22
main()
{
int count, total = 0;
for(count =1; count <= NUMBER; count++, total += count)
printf("Vive le langage C !!!\n");
printf("Le total est %d\n", total);
}
Initialisation
Condition de fin de boucle
Incrémentation et autres fonctions

Boucle « while »

```
/* Boucle while */
#include <stdio.h>
#define NUMBER 22
main()
{
   int count = 1, total = 0;

   while(count <= NUMBER)
   {
      printf("Vive le langage C !!!\n");
      count++;
      total += count;
   }
   printf("Le total est %d\n", total);
}
```



Boucle « do while »

```
/* Boucle do while */
#include <stdio.h>
#define NUMBER 22
main()
{
   int count = 1, total = 0;

   do
   {
      printf("Vive le langage C !!!\n");
      count++;
      total += count;
   }  while(count <= NUMBER);
   printf("Le total est %d\n", total);
}
```

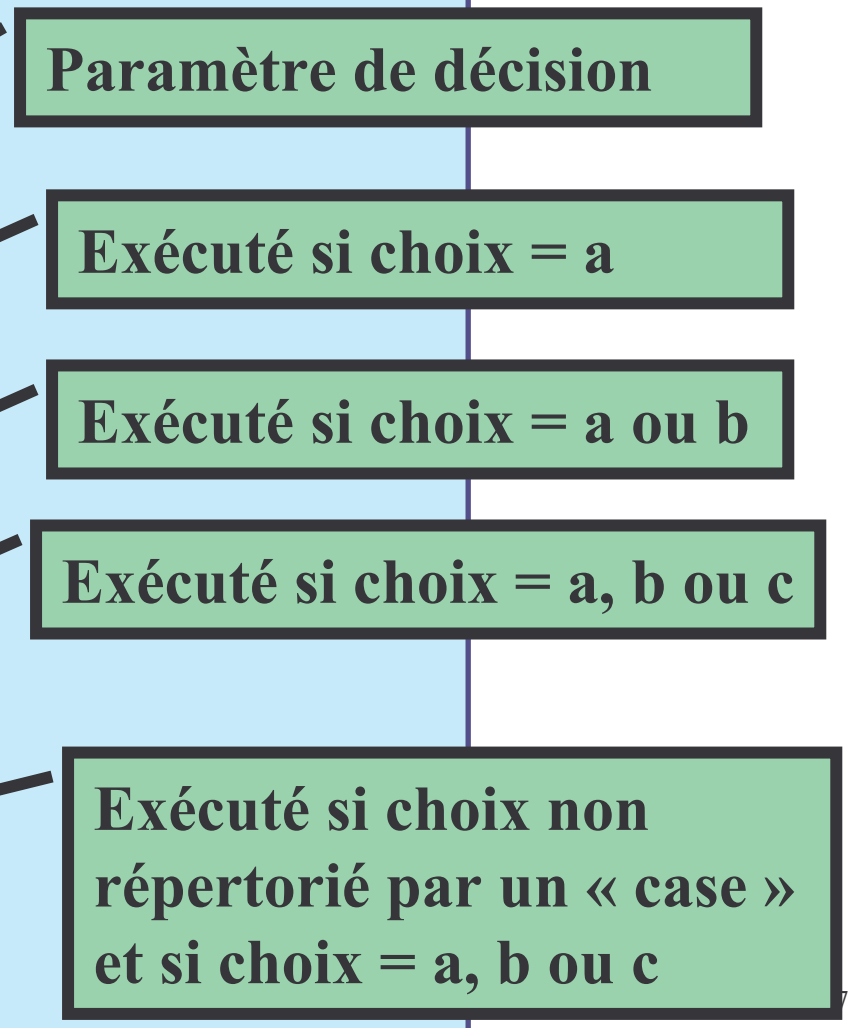
Initialisation

Incrémentation

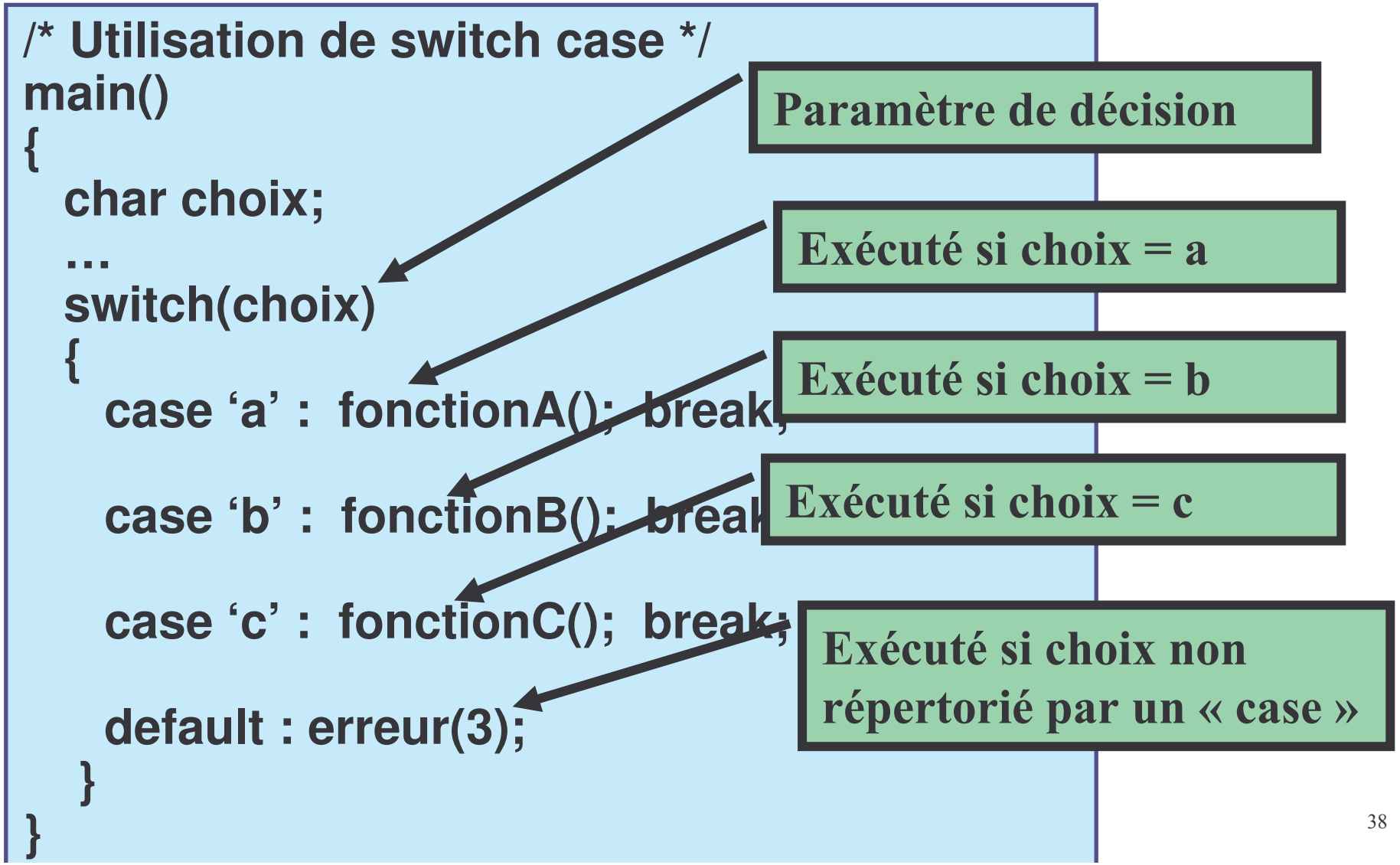
Condition de fin de boucle
(boucle tant que vrai)
(*boucle faite au moins 1 fois*)

Choix multiple: « switch case »

```
/* Utilisation de switch case */
main()
{
   char choix;
   …
   switch(choix)
   {
      case ‘a’ :  fonctionA();

      case ‘b’ :  fonctionB();

      case ‘c’ :  fonctionC();

      default : erreur(3);
    }
}
```



Effet du « break »



Directives

- #define <name> <substitution>
  - #define ZERO 0
- #undefine <name>
  - #undefine ZERO
- #include <filename>
  - #include <math.h>
- #if, #endif, #else

Modèle de la mémoire

- Le compilateur C génère 6 sections (ou blocs) de code et de données relocalisables.

Modèle de la mémoire

- **Sections initialisées:**
  - .text:
    - code exécutable.
  - .cinit:
    - table des variables globales et statiques initialisées.
  - .const:
    - table des valeurs d'initialisation des constantes globales et statiques + les chaînes de car.

Modèle de la mémoire

- **Sections non-initialisées:**
  - .bss:
    - espace réservé pour les variables globales et statiques non initialisées.
  - .stack:
    - pile système.
  - .sysmem:
    - pool de mémoire pour allocation dynamique (alloc, malloc, calloc).

Modèle de la mémoire

- La section « .data » n'est pas utilisée par le compilateur C (réservé pour l'assembleur).