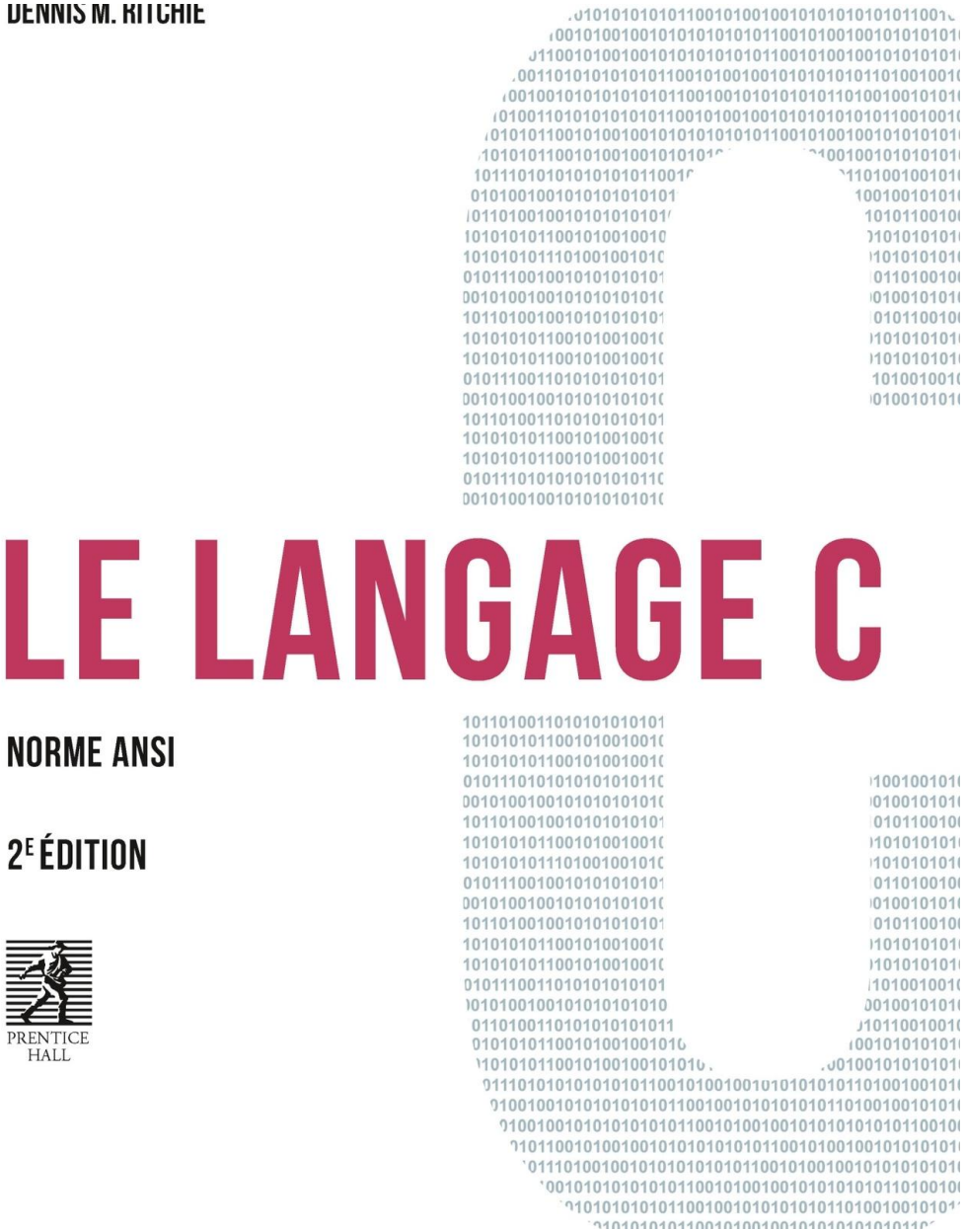
```c
int number1, number2, sum;

printf("Enter First Number: ");
scanf("%d", &number1);

printf("Enter Second Number: ");
scanf("%d", &number2);

// calculating sum
sum = number1 + number2;

printf("\nAddition of %d and %d is %d", number1, number2, sum);
```

Introduction au Langage C

Langage de programmation C

Safwan.Chendeb@lametice.com



Historique et caractéristiques du langage C

Le langage C a été développé par Dennis Ritchie chez Bell Labs entre 1972 et 1973 pour le système d'exploitation Unix.

Ses principales caractéristiques sont la concision, l'efficacité, la portabilité et la stabilité, ce qui en fait un outil incontournable dans de nombreux domaines.

Introduction au Langage C

Historique du langage C

Le langage C a été développé dans les années 1970 par Dennis Ritchie chez les Laboratoires Bell.

Domaines d'application

Le langage C est largement utilisé dans le développement de systèmes d'exploitation, de programmes système, de pilotes de périphériques et d'applications hautes performances.

Caractéristiques principales

C'est un langage de programmation impératif, compilé, portable, de bas niveau et offrant un contrôle très fin de la mémoire.

Flexibilité et efficacité

Le langage C permet de mélanger du code de haut niveau et de bas niveau, offrant ainsi une grande flexibilité et une excellente performance.

Le langage C est un langage de programmation essentiel, doté d'un riche héritage et largement utilisé dans de nombreux domaines. Sa combinaison de bas niveau et de haut niveau en fait un outil puissant et polyvalent pour les développeurs.

C Language History

1972-1973

Developed by Dennis Ritchie at Bell Labs

Stable and Portable

Considered a high-level assembly language

Widespread Use

Used to implement other programming languages

Efficient Code Generation

C compilers can generate highly optimized machine code

Jargon du programmeur C

- **Code source**

  Ce que vous tapez sur la machine. Le programme que vous écrivez.

- **Compiler (build)**

  À partir du code source, construire un programme compréhensible par une machine.

- **Exécutable**

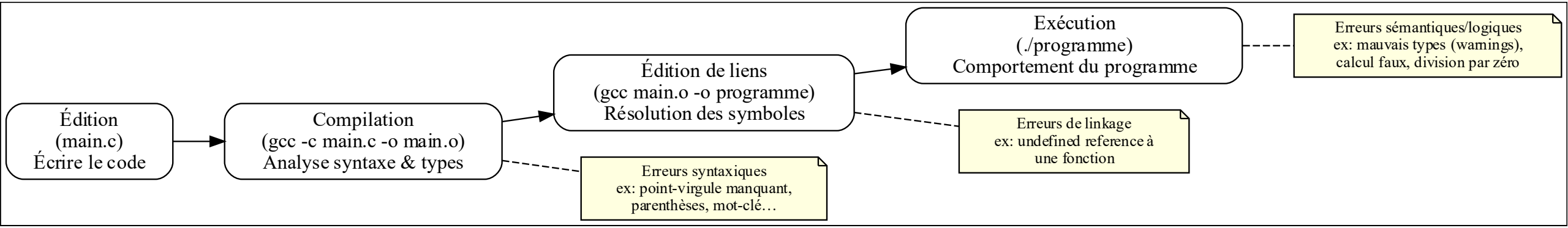
  Le programme compilé qu'une machine peut exécuter.

- **Bibliothèque**

  Fonctions prédéfinies du langage C.

- **Fichier entête (.h)**

  Fichiers ayant l'extension .h qui sont inclus au début du code source pour déclarer des fonctions et constantes.



Règles fondamentales du langage C

- Terminaison des instructions

  Toute instruction se termine par un point-virgule '**;**'.

- Sensibilité à la casse (majuscule miniscule)

  Le langage C est sensible à la casse, c'est-à-dire que int est différent de INT. Tous les mots-clés en C sont en minuscules.

- Lisibilité du code

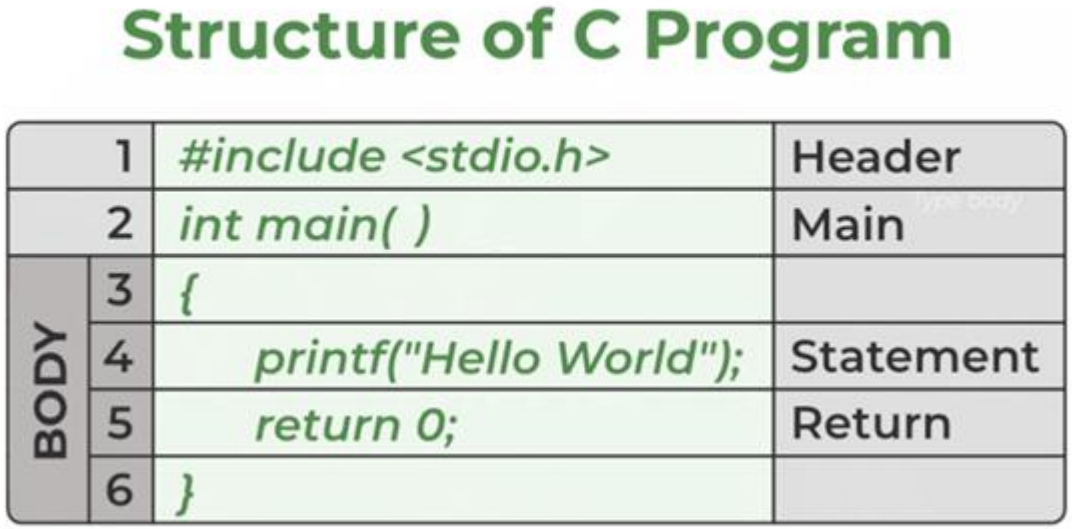
  L'indentation est importante pour la lisibilité du code, bien que les espaces et les indentations soient ignorés par le compilateur.

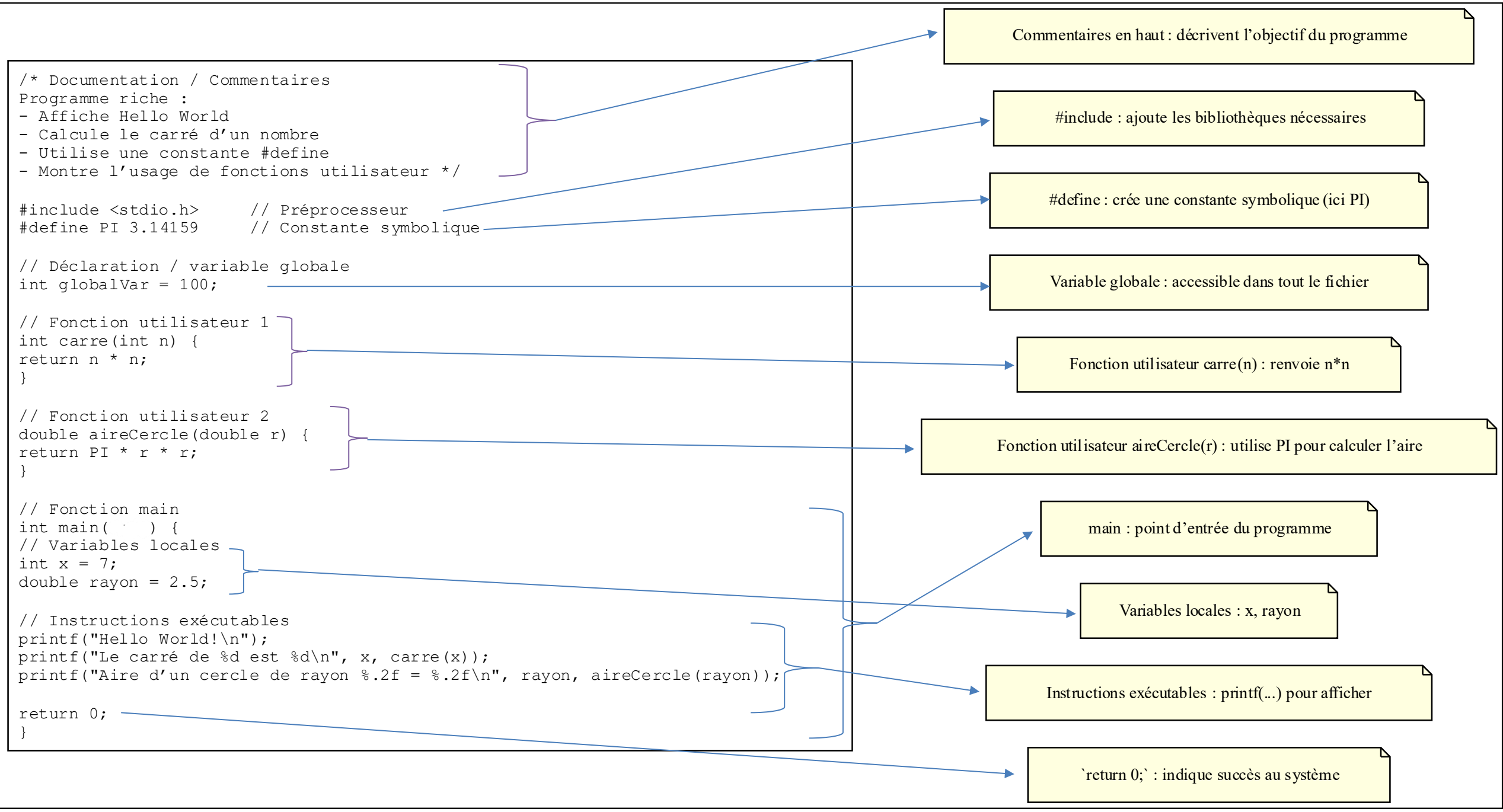
- Représentation des retours à la ligne

  Un retour à la ligne est représenté par **\n**.

Structure d'un programme C

Chaque programme C a un point d'entrée unique : la fonction main. L'unique porte d'entrée de tout programme C est la fonction main. Les instructions du programme sont placées dans le bloc de la méthode main, délimité par des accolades {}. La fonction main se termine systématiquement par l'instruction return 0;.




/* Documentation / Commentaires
Programme riche :
- Affiche Hello World
- Calcule le carré d'un nombre
- Utilise une constante #define
- Montre l'usage de fonctions utilisateur */
#include <stdio.h> // Préprocesseur
#define PI 3.14159 // Constante symbolique
// Déclaration / variable globale
int globalVar = 100;
// Fonction utilisateur 1
int carre(int n) {
return n * n;
}
// Fonction utilisateur 2
double aireCercle(double r) {
return PI * r * r;
}
// Fonction main
int main( ) {
// Variables locales
int x = 7;
double rayon = 2.5;
// Instructions exécutables
printf("Hello World!\n");
printf("Le carré de %d est %d\n", x, carre(x));
printf("Aire d'un cercle de rayon %.2f = %.2f\n", rayon, aireCercle(rayon));
return 0;
}
Commentaires en haut : décrivent l'objectif du programme
#include : ajoute les bibliothèques nécessaires
#define : crée une constante symbolique (ici PI)
Variable globale : accessible dans tout le fichier
Fonction utilisateur carre(n) : renvoie n*n
Fonction utilisateur aireCercle(r) : utilise PI pour calculer l'aire
main : point d'entrée du programme
Variables locales : x, rayon
Instructions exécutables : printf(...) pour afficher
`return 0;` : indique succès au système

Les commentaires en C

Commentaires sur une seule ligne

Précédés par **//**, les commentaires sur une seule ligne permettent d'ajouter des notes explicatives au code.

Commentaires de bloc

Entourés par **/*** et ***/**, les commentaires de bloc offrent la possibilité de commenter des portions de code plus importantes.

Utilité des commentaires

Les commentaires permettent d'ajouter des informations pour mieux comprendre le code ou de désactiver temporairement certaines parties du code en les excluant de la compilation.

Les commentaires, qu'ils soient sur une seule ligne ou sous forme de bloc, sont des éléments essentiels du langage C pour améliorer la lisibilité et la compréhension du code.

Variables et types de données

Définition des variables

Les variables sont des conteneurs permettant de stocker des valeurs. Elles peuvent contenir différentes valeurs au cours de l'exécution du programme.

Types de données courants

Les types de données les plus fréquents en C sont : **char** (caractère), **int** (entier), **float** (nombre à virgule flottante) et **double** (nombre à virgule flottante double précision).

Déclaration des variables

Une variable est définie par son type et son nom. Plusieurs variables du même type peuvent être déclarées sur une même ligne, séparées par des virgules.

Initialisation des variables

Une variable peut être initialisée lors de sa déclaration avec une valeur de départ. Il est également possible de modifier la valeur d'une variable après sa déclaration.

Tailles des types de données

Chaque type de données occupe un espace mémoire spécifique : char (1 octet), int (4 octets), float (4 octets) et double (8 octets).

Manipulation des variables

- ## Définition de variables

  Déclarer le type et le nom de la variable, par exemple : int i;

- ## Initialisation de variables

  Affecter une première valeur à la variable, par exemple : i = 3; ou int i = 3;

```
// --- Initialisation de variables ---
int i;
i = 3; // On donne une première valeur à i
int j = 10; // Déclaration ET initialisation en une seule ligne
```

- ## Modification de variables

  Changer la valeur d'une variable, par exemple : i = 5; ou i = i * 2;

```
// --- Modification de variables ---
i = 5; // On change la valeur de i
i = i * 2; // On multiplie i par 2 (sa valeur devient 10)
```

- ## Déclaration de plusieurs variables

  Déclarer plusieurs variables de même type sur une même ligne, séparées par des virgules

```
// --- Déclaration de plusieurs variables ---
int a = 1, b = 2, c = 3; // Plusieurs entiers déclarés et initialisés
```

```c
#include <stdio.h>

// Définition de constantes symboliques
#define PI 3.14159
#define MESSAGE "Bonjour, constantes en C !"

int main(void) {
    double rayon = 5.0;
    double aire = PI * rayon * rayon; // Utilisation de la
constante PI

    printf("%s\n", MESSAGE);          // Utilisation de la
constante MESSAGE
    printf("Aire du cercle = %.2f\n", aire);

    return 0;
}
```

Les constantes en C

Les constantes en C sont des valeurs fixes qui ne peuvent pas changer pendant l'exécution du programme. Elles sont définies à l'aide de la directive **#define**, permettant d'attribuer une valeur numérique ou textuelle à un nom symbolique. Cette approche facilite la maintenance et la compréhension du code, en évitant les valeurs « magiques » dispersées dans le programme.

Conventions de nommage

Variables significatives

Utiliser des noms de variables qui décrivent clairement leur rôle et leur fonction dans le programme.

Séparation des mots

Utiliser des majuscules ou des tirets de soulignement (_) pour séparer les mots dans les noms de variables.

Respect de la casse

Respecter la sensibilité à la casse du langage C, en utilisant les minuscules pour les mots-clés et les majuscules pour les constantes.

Le respect des conventions de nommage en C est essentiel pour assurer la lisibilité et la maintenabilité du code. En utilisant des noms significatifs, une séparation appropriée des mots et en respectant la sensibilité à la casse, vous contribuerez à rendre votre code plus compréhensible et plus facile à travailler.

Conventions de nommage

Catégorie	Convention	Exemple
Fichiers	lower_snake_case.c / .h	mon_module.c
Gardiens d'en-tête	UPPER_SNAKE_CASE	DEMO_MODULE_H
Fonctions	lower_snake_case()	calc_area(), read_config()
Variables	lower_snake_case	buffer_len, line_count
Macros / #define	UPPER_SNAKE_CASE	MAX_SIZE, DEFAULT_PATH
Types (struct/enum)	PascalCase	Point, Color
Champs de struct	lower_snake_case	x, y
Valeurs enum	UPPER_SNAKE_CASE	COLOR_RED, COLOR_GREEN
Préfixes utiles	g_ globales | s_ static	

```c
#define MAX_POINTS 1024
typedef struct {
   double x;
   double y;
}Point;

typedef enum { COLOR_RED, COLOR_GREEN, COLOR_BLUE } Color;
void calc_area(double radius);
```

Les Opérateurs

- Les opérateurs arithmétiques

  Permettent d'effectuer des opérations mathématiques de base comme l'addition, la soustraction, la multiplication et la division. +,-,*,/, %

- Les opérateurs d'affectation

  Permettent d'assigner une valeur à une variable et de chaîner plusieurs opérations en une seule instruction. =, +=, -=, *=, /=

- Les opérateurs d'incrémentation

  Permettent d'augmenter ou de diminuer d'une unité la valeur d'une variable de manière simplifiée. ++,--

- Les opérateurs relationnels

  Permettent d'effectuer des comparaisons entre deux variables et de vérifier leur égalité, leur infériorité ou leur supériorité. ==, !=, <, <=, >, >=

- Les opérateurs logiques

  Permettent d'évaluer la véracité de plusieurs conditions en utilisant les opérateurs 'et', 'ou' et 'non’. &&, ||, !

Les opérateurs en C

Opérateur	Description	Exemple
+ (Addition)	Effectue l'addition de deux opérandes	a + b
- (Soustraction)	Effectue la soustraction de deux opérandes	a - b
* (Multiplication)	Effectue la multiplication de deux opérandes	a * b
/ (Division)	Effectue la division de deux opérandes	a / b
% (Modulo)	Retourne le reste de la division de deux opérandes	a % b
=	Affecte la valeur de l'opérande de droite à l'opérande de gauche	a = 5
==, !=, <, <=, >, >=	Opérateurs de comparaison, renvoient 1 si la condition est vraie, 0 sinon	a == b, a != b, a < b, a > b, a <= b, a >= b
&&, ||, !	Opérateurs logiques (ET, OU, NON)	a && b, a || b, !a
++, --	Incrémentation/Décrémentation de 1	a++, a--
+=, -=, *=, /=	Opérateurs d'affectation combinée	a += b, a -= b, a *= b, a /= b

Incrementation/Decrementation

Post incrementation

```c
int a;
a=5;
a++;
printf("a=%d",a); // a=6
```

```c
int a,b,c;
a=5;
b=a++;
c=a;
printf("a=%d,b=%d,c=%d",a,b,c); //a=6,b=5,c=6
```

Pré incrementation

```c
int a;
a=5;
++a;
printf("a=%d",a); // a=6
```

```c
int a,b,c;
a=5;
b=++a;
c=a;
printf("a=%d,b=%d,c=%d",a,b,c); //a=6,b=6,c=6
```

Les caractères et les chaînes de caractères

- Caractères seuls

  Les caractères seuls sont représentés par de simples quotes ('), comme ‘A’. (char a= ‘A’;)

- Définition d'une chaîne de caractères

  Une chaîne de caractères est une suite de caractères représentée par des guillemets doubles. **”Bonjour”**

- Déclaration d'une chaîne de caractères

  Les chaînes de caractères sont déclarées à l'aide du type de données char* ou string. **char * s;**

- Initialisation d'une chaîne de caractères

  Une chaîne de caractères peut être initialisée avec une valeur constante entre guillemets doubles. char * s=“Hello”;

- Manipulation des chaînes de caractères

  Les chaînes de caractères peuvent être concaténées, copiées, comparées et modifiées à l'aide de fonctions prédéfinies.

- Lecture et écriture de chaînes de caractères

  Les fonctions scanf() et printf() permettent respectivement de lire et d'écrire des chaînes de caractères.

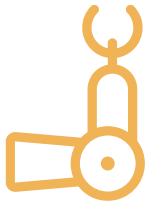
- Différence entre simples et doubles quotes

  Les simples quotes sont utilisées pour les caractères seuls, tandis que les doubles quotes sont utilisées pour les chaînes de caractères.

Conversions Implicites et Explicites

Définition des conversions de types

Le fait de modifier le type d'une donnée en un autre type.



Conversions implicites

Modification du type de données effectuée automatiquement par le compilateur sans retourner d'erreur.

```c
int x;
double y;
x = 8.324;
y = 4;
```

Conversions explicites (cast)

Modification du type de données forcée par l'utilisateur en mettant le type souhaité entre parenthèses.

Exemples de conversions

Conversion d'un entier en réel, d'un réel en entier, etc.

```c
int x;
double y;
x = (int)2.52;
y = (double)1;
printf("x = %d \ny = %f", x,y);
```

```
x = 2
y = 1.000000
```

Les conversions de types, qu'elles soient implicites ou explicites, sont des opérations essentielles en programmation C pour garantir la compatibilité entre les différentes données manipulées dans un programme.

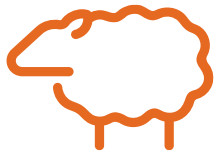
Le Type Booléen

Définition du type booléen

Le langage C ne définit pas de type booléen par défaut, mais il est possible de le simuler.

Valeurs booléennes

La valeur nulle (0) représente la constante booléenne faux (false), tandis que toute autre valeur est assimilée à la constante vrai (true).



Création d'un type pseudo-booléen

En utilisant la directive #define, il est possible de créer un type booléen avec les constantes TRUE et FALSE.

```
#define BOOL int
#define TRUE 1
#define FALSE 0
```

Utilisation intuitive des booléens

Les variables de type booléen peuvent être affectées directement avec les constantes TRUE et FALSE.

Le langage C permet de travailler avec des données booléennes, même s'il ne définit pas de type booléen par défaut. En utilisant des directives de préprocesseur, il est possible de simuler un type booléen et de l'utiliser de manière intuitive dans son code.

Les Entrées/Sorties

Interaction avec l'utilisateur

Les entrées/sorties permettent d'afficher des informations à l'écran et de récupérer des données saisies par l'utilisateur.

Fonctions essentielles

Les fonctions printf() et scanf() sont indispensables pour ces opérations d'entrées/sorties.

Affichage formaté

printf() permet d'effectuer un affichage formaté à l'écran en utilisant des formats spécifiques pour les différents types de données.

Saisie de données

scanf() permet de demander à l'utilisateur de fournir des informations en utilisant également des formats spécifiques pour les différents types de données.

Les entrées/sorties en langage C sont des éléments fondamentaux pour la communication entre le programme et l'utilisateur. Grâce aux fonctions printf() et scanf(), les développeurs peuvent facilement afficher des informations à l'écran et récupérer des données saisies par l'utilisateur, rendant ainsi possible l'interaction avec l'utilisateur final.

Les Entrées/Sorties

printf() : Affichage à l'écran

Fonction prédéfinie de la bibliothèque standard stdio.h permettant d'effectuer un affichage à l'écran de manière formatée.

scanf() : Saisie de données

Fonction permettant de demander à l'utilisateur de fournir des informations, avec gestion du format des données saisies.

Formats de données

Explication des différents formats de données pris en charge par printf() et scanf() (%d, %f, %c, %s, etc.) pour un affichage/saisie personnalisés.

Exemples d'utilisation

Présentation d'exemples concrets d'utilisation de printf() et scanf() dans un programme C.

Les fonctions printf() et scanf() constituent les outils de base pour l'affichage à l'écran et la saisie de données en langage C. Leur maîtrise est essentielle pour la création de programmes interactifs et fonctionnels.

Affichage à l'Écran avec printf()

Fonction printf()

La fonction printf() permet d'effectuer un affichage formaté à l'écran.

Inclusion de stdio.h

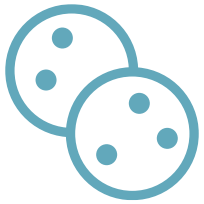
L'utilisation de printf nécessite la présence de #include

Formats spécifiques

printf utilise des formats spécifiques pour les différents types de données comme %d pour les entiers, %f pour les réels, etc.

La fonction printf() est un outil essentiel pour l'affichage de données formatées à l'écran en langage C. Elle permet une interaction efficace entre le programme et l'utilisateur.

Formatage des Sorties



Types de données formatés

Présentation des différents formats de données pris en charge par printf() comme %d pour les entiers, %f pour les flottants, %c pour les caractères, %s pour les chaînes de caractères, etc.

Contrôle de la largeur d'affichage

Utilisation de la syntaxe %Xd/%Xf pour spécifier la largeur minimale d'affichage d'un champ, X étant le nombre de caractères à réserver.

Contrôle de la précision d'affichage

Utilisation de la syntaxe %.Xf pour définir le nombre de chiffres après la virgule pour l'affichage des nombres flottants, X étant le nombre de chiffres après la virgule.

Combinaison des options de formatage

Possibilité de combiner les options de largeur et de précision pour un affichage encore plus personnalisé, par exemple %10.2f pour un flottant sur 10 caractères avec 2 chiffres après la virgule.

Le formatage des sorties avec printf() offre de nombreuses possibilités pour personnaliser l'affichage des données en fonction des besoins. La maîtrise de ces options de formatage est essentielle pour produire des sorties claires et lisibles.

```c
#include <stdio.h>

int main(void) {
    int entier = 42;
    double pi = 3.14159265;
    char car = 'A';
    char *mot = "Bonjour";

    // Affichage d'un entier
    printf("Entier : %d\n", entier);
    // Affichage d'un entier avec largeur fixe
    printf("Entier aligné sur 5 caractères : %5d\n", entier);
    // Affichage en hexadécimal
    printf("Entier en hexadécimal : %x\n", entier)
    // Affichage d'un flottant
    printf("Pi = %f\n", pi);
    // Affichage d'un flottant avec 2 décimales
    printf("Pi (2 décimales) = %.2f\n", pi);
    // Affichage d'un caractère
    printf("Caractère : %c\n", car);
    // Affichage d'une chaîne de caractères
    printf("Texte : %s\n", mot);
    return 0;
}
```

```
Entier : 42
Entier aligné sur 5 caractères :    42
Entier en hexadécimal : 2a
Pi = 3.141593
Pi (2 décimales) = 3.14
Caractère : A
Texte : Bonjour
```

Exemples de printf()

La fonction printf() en langage C permet d'afficher des informations à l'écran de manière formatée. Elle offre une grande flexibilité dans l'affichage de différents types de données, tels que les entiers, les nombres réels, les caractères et les chaînes de caractères.

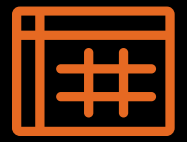
Saisie de Données avec scanf()

Fonction scanf()

Permet de demander à l'utilisateur de fournir des informations.

Formats spécifiques

Utilise des formats spécifiques pour les différents types de données (%d, %f, %c, etc.)



Stockage dans des variables

Les valeurs saisies sont stockées dans des variables, précédées du caractère &, sauf pour les chaînes de caractères.

Nécessite #include

L'utilisation de scanf() nécessite l'inclusion de la bibliothèque standard stdio.h.

La fonction scanf() est essentielle pour permettre à l'utilisateur de fournir des données à un programme en langage C. Elle offre une grande flexibilité dans la saisie de différents types de données, tout en les stockant dans des variables pour une utilisation ultérieure.

```c
#include <stdio.h>

int main(void) {
    int age;
    double pi;
    char lettre;
    char mot[20];   // assez grand pour le mot + '\0'

    printf("Age ? ");
    scanf("%d", &age);

    printf("Pi (double) ? ");
    scanf("%lf", &pi);

    // L'espace avant %c consomme
    // les blancs/retours de ligne restants
    printf("Lettre ? ");
    scanf(" %c", &lettre);

    // Limitation à 19 caractères pour éviter le dépassement
    printf("Un mot (<= 19 chars) ? ");
    scanf("%19s", mot);

    printf("\n--- Résumé ---\n");
    printf("age=%d\n", age);
    printf("pi=%.4f\n", pi);
    printf("lettre=%c\n", lettre);
    printf("mot=%s\n", mot);

    return 0;
}
```

Exemples de scanf()

La fonction scanf() permet de récupérer des données saisies par l'utilisateur. Voici quelques exemples de son utilisation pour lire différents types de données en langage C.

```
Age ? 21
Pi (double) ? 3.14159
Lettre ? A
Un mot (<= 19 chars) ?
Bonjour
--- Résumé ---
age=21
pi=3.1416
lettre=A
mot=Bonjour
```

Options de Formatage pour printf() et scanf()

Format	Type de Données	Description
%d	Entier décimal	Affiche ou lit un nombre entier signé dans le système décimal
%ld	Entier décimal long	Affiche ou lit un nombre entier long signé dans le système décimal
%x	Entier hexadécimal	Affiche ou lit un nombre entier dans le système hexadécimal
%c	Caractère	Affiche ou lit un caractère unique
%s	Chaîne de caractères	Affiche ou lit une chaîne de caractères
%f	Nombre réel	Affiche ou lit un nombre réel à virgule flottante
%lf	Nombre réel double	Affiche ou lit un nombre réel à virgule flottante double précision

Les structures de contrôle en Langage C

Structures de contrôle en Langage C

Présentation des principales structures de contrôle en Langage C, notamment les structures conditionnelles et les boucles.

Structures conditionnelles

Explication des instructions if, if-else et switch-case et leur utilisation dans des programmes en Langage C.

Boucles

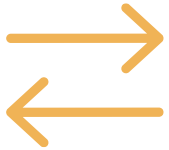
Présentation des différents types de boucles (for, while, do-while) et leur utilisation dans la programmation en Langage C.

Cette introduction présente les principales structures de contrôle en Langage C, qui sont essentielles pour créer des programmes efficaces et robustes.

Les Structures Conditionnelles

Structures conditionnelles if-else

Permet de choisir entre deux blocs d'instructions en fonction d'une condition.



Structures conditionnelles switch-case

Permet de choisir parmi plusieurs blocs d'instructions en fonction de la valeur d'une variable.

Utilisation des opérateurs relationnels

Les opérateurs tels que <, >, <=, >=, == et != sont utilisés pour construire les conditions.

Utilisation des opérateurs logiques

Les opérateurs logiques tels que &&, || et ! permettent de combiner plusieurs conditions.

Les structures conditionnelles if-else et switch-case sont des outils essentiels pour prendre des décisions dans un programme en fonction de certaines conditions. Elles permettent de créer des programmes plus flexibles et adaptables aux différents scénarios.

Structures conditionnelles

if simple

```c
int x = 7;
if (x > 5) {
    printf("x est strictement supérieur à 5\n");
}
```

Opérateur ternaire (? :)

```c
int age = 20;
char *statut = (age >= 18) ? "majeur" : "mineur";
printf("Statut: %s\n", statut);
```

switch / case

```c
int note = 14;
switch (note) {
    case 20: case 19: case 18: case 17: case 16:
        printf("Très bien\n");
        break;
    case 15: case 14: case 13: case 12:
        printf("Assez bien\n");
        break;
    case 11: case 10:
        printf("Passable\n");
        break;
    default:
        printf("Insuffisant\n");
        break;
}
```

if avec alternative (if / else)

```c
int x = 3;
if (x % 2 == 0) {
    printf("x est pair\n");
}
else {
    printf("x est impair\n");
}
```

if avec plusieurs alternatives (if / else if / else)

```c
int note = 14;
if (note >= 16) {
    printf("Très bien\n");
}
else {
    if (note >= 12) {
        printf("Assez bien\n");
    }
    else {
        if (note >= 10) {
            printf("Passable\n");
        }
        else {
            printf("Insuffisant\n");
        }
    }
}
```

Les Structures Répétitives

Boucle while

Utilisation de la boucle while pour répéter un bloc d'instructions tant qu'une condition est vraie.

(verifier la condition en premier lieu)

Boucle for

Utilisation de la boucle for pour répéter un bloc d'instructions un nombre de fois défini.

(pas que !)

Boucle do-while

Utilisation de la boucle do-while pour exécuter un bloc d'instructions au moins une fois, puis le répéter tant qu'une condition est vraie.

Instructions break et continue

Utilisation des instructions break et continue pour contrôler le déroulement des boucles.

NON recommandé

Les structures répétitives en C, telles que les boucles while, for et do-while, permettent d'exécuter un bloc d'instructions de manière itérative. Les instructions break et continue offrent un contrôle supplémentaire sur le déroulement des boucles. Ces structures sont essentielles pour la programmation en C et permettent de résoudre de nombreux problèmes de manière efficace.

Boucle for

La boucle for permet de répéter un bloc d'instructions **selon trois étapes bien définies** :

```
for (initialisation; condition; action) {
    // instructions à répéter
}
```

Initialisation : se fait une seule fois, avant le début de la boucle. (Ex : int i = 0;)

Condition : test logique évalué avant chaque itération. Si elle est vraie, on exécute le corps de la boucle ; sinon, on sort.

Action : exécutée à la fin de chaque tour de boucle, pour mettre à jour les variables de contrôle. (Ex : i++, i *= 2, j--, ...)

```
for (int i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
```

```
for (int i = 1; i < 1000; i *= 2) {
    printf("%d\n", i);
}
```

```
int tab[4] = {5, 8, 12, 20};
int taille = 4;

for (int i = 0; i < taille; i++) {
    printf("Élément %d : %d\n", i, tab[i]);
}
```

```
char mot[8] = "Bonjour";
//Le '\0' est automatiquement ajouté

for (int i = 0; i < 7; i++) {
    printf("Lettre : %c\n", mot[i]);
}
```

Boucle while

- **La boucle while** permet de répéter un bloc d'instructions **tant qu'une condition est vraie**.

  Utilisée lorsque le nombre d'itérations n'est pas connu mais que l'on dispose d'une condition qui fera arrêter la boucle après un nombre de tours donné. La boucle while vérifie la condition avant d'exécuter le bloc d'instructions.
  C'est donc une **boucle à entrée conditionnelle**.

```c
while (condition) {
    // instructions à répéter
}
```

```c
int i = 1;

while (i <= 10) {
    printf("%d\n", i);
    i++;
}
```

```c
char mot[] = "Bonjour";
int i = 0;

while (mot[i] != '\0') {
    printf("Lettre %d : %c\n", i, mot[i]);
    i++;
}
```

```c
int valeur = 0;

while (valeur != 42) {
    printf("Entrez un nombre : ");
    scanf("%d", &valeur);
}
```

Boucle do-while

- **La boucle do...while** permet d'**exécuter** un bloc d'instructions **puis le répéter tant qu**'une condition est vraie, le test est effectué après l'exécution du bloc.

  Utilisée lorsque l'on souhaite exécuter la première itération sans condition et faire soumettre le reste des itérations à une ou plusieurs conditions. La boucle do-while exécute d'abord le bloc d'instructions puis vérifie la condition. C'est donc une **boucle à sortie conditionnelle.**

```
//Syntaxe de base
do {
    // instructions
} while (condition);
```

```
//Lire jusqu'à saisir une valeur valide
int valeur;

do {
    printf("Entrez un nombre (entre 1 et 10) : ");
    scanf("%d", &valeur);
} while (valeur < 1 || valeur > 10);
```

```
//Afficher un menu au moins une fois
char choix;

do {
    printf("Menu :\n");
    printf(" a) Option A\n b) Option B\n q) Quitter\n");
    printf("Votre choix : ");
    scanf(" %c", &choix);
} while (choix != 'q');
```

Break vs Continue

Définitions

- break : Mot clé utilisé seulement dans l'instruction switch ou dans une boucle. Permet de quitter le switch ou la boucle la plus proche, pour exécuter l'instruction suivante.
- continue : Mot clé utilisé seulement dans une boucle. Permet de quitter l'itération courante d'une boucle et de passer à l'itération suivante.

Utilisation appropriée

- break : Permettre de sortir d'une boucle ou d'un switch lorsqu'une condition particulière est remplie. (très déconseillé pour la boucle)
- continue : Permettre de passer à l'itération suivante d'une boucle lorsqu'une condition particulière est remplie, sans exécuter le reste du bloc de la boucle.

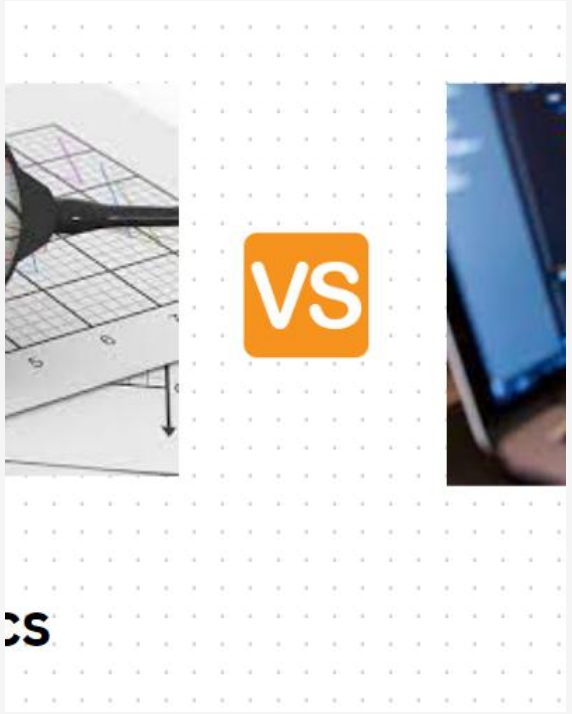
Exemple avec break

```
switch (mois) {
    case 1:
        printf("Janvier");
        break;
    case 2:
        printf("Fevrier");
        break;
    default :
        printf("Ce numC)ro de mois n'existe pas");
    }
```

Exemple avec continue

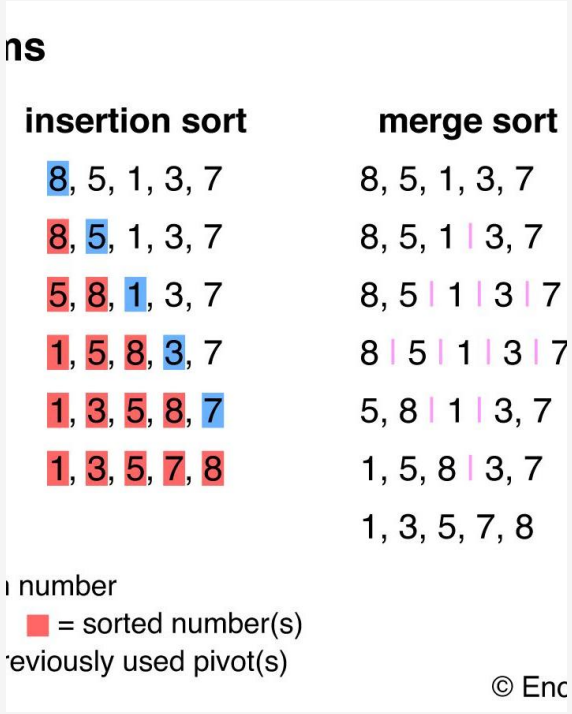
```
for (int x = 0; x < 10; x++) {
        if (x % 2 == 0) {
            continue;
        }
        printf("%d ", x);
   }
```

Cas d'utilisation



Traitement des données

Utilisation des structures de contrôle pour analyser et manipuler des données complexes dans des programmes en Langage C.

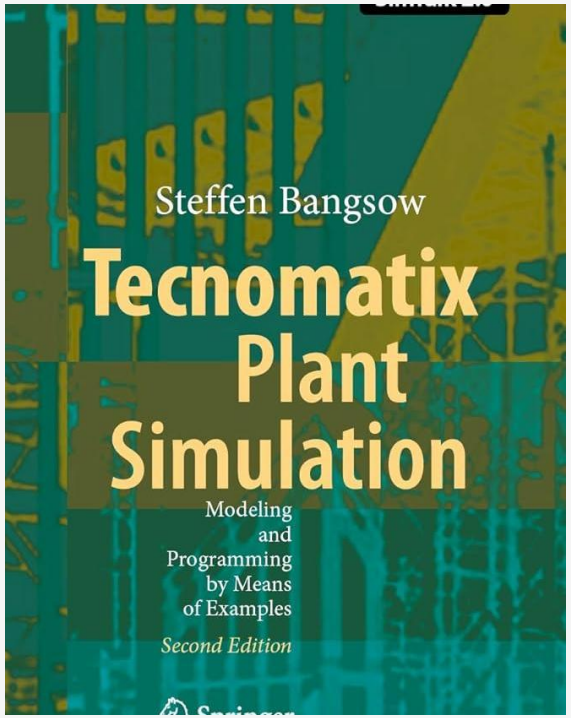


Algorithmes de tri

Implémentation efficace d'algorithmes de tri en utilisant les boucles et les conditions en Langage C.

Gestion des erreurs

Utilisation des structures conditionnelles pour détecter et gérer les erreurs dans des programmes en Langage C.



Simulations et modélisations

Création de simulations et de modèles complexes en Langage C à l'aide des structures de contrôle.

Conclusion

Résumé des principaux concepts abordés

Nous avons abordé les bases fondamentales du langage C, notamment les variables, les types, les opérateurs, les conversions, les entrées/sorties et les structures de contrôle.

Perspectives d'apprentissage complémentaires

Pour approfondir vos connaissances en programmation C, vous pouvez explorer des sujets tels que la gestion de la mémoire, les structures de données, la programmation modulaire et la gestion des erreurs.

Cette présentation vous a permis d'acquérir une compréhension solide des concepts clés du langage C. Continuez à pratiquer et à explorer davantage pour devenir un développeur C accompli.

Le langage C est un outil puissant et polyvalent qui a façonné de nombreux systèmes et applications modernes. Grâce à son efficacité, sa portabilité et sa stabilité, il reste un langage incontournable dans de nombreux domaines, de l'embarqué aux systèmes d'exploitation. Cette présentation a permis de vous familiariser avec les concepts fondamentaux du C, vous donnant les bases nécessaires pour explorer davantage ses capacités et vous lancer dans la programmation.