Chapitre 1: Notion langage c

Langage

,

I. Notion Langage C
   1. Programme
   2. Processus

II. Structure Langage C
   1. Type
   2. Identificateur
   3. Constantes
   4. Variables
   5. syntaxes déclaration

III. Opérateurs
   1. Arithmétique
   2. Comparaison
   3. Logique
   4. Affectation
   5. Notion d’expression

I. Notion Langage C

Langage

1. Programme

- Directives du préprocesseur:
  Les directives du préprocesseur commencent par **# .**
  Elles sont utilisées pour inclure des fichiers, définir des macros et conditionner la compilation de certaines parties du code.

- Déclaration des variables:
  Les variables doivent être déclarées avant d'être utilisées.
  La déclaration indique le type de données et le nom de la variable.

- Fonction principale:
  Tout programme C doit avoir une fonction main.
  C'est le point d'entrée du programme.

- Instructions:
  Les instructions sont les éléments constitutifs du programme.
  Elles peuvent être des affectations, des appels de fonctions, des instructions conditionnelles, des boucles, etc.

- Commentaires
  Les commentaires sont utilisés pour expliquer le code.
  Ils sont ignorés par le compilateur.

I. Notion Langage C

1. Programme

- Exemple de programme simple en C:

```c
#include <stdio.h>

int main() {
  int a = 10;
  int b = 20;

  // Addition de a et b
  int c = a + b;

  // Affichage du résultat
  printf("La somme de a et b est : %d\n", c);

  return 0;
}
```

I. Notion Langage C

2. Processus

- Écriture du Code Source (IDE) :

Le processus commence par l'écriture du code source en langage C. Le code est rédigé dans un éditeur de texte ou un environnement de développement intégré (IDE). Il doit suivre la syntaxe spécifique du langage C, avec des déclarations de variables, des fonctions, des structures de contrôle, etc.

- Compilation

Le code source C est écrit dans un fichier texte avec l'extension .c.
Le compilateur C traduit le code source en langage machine que l'ordinateur peut comprendre.
Le résultat de la compilation est un fichier exécutable avec l'extension .exe (sous Windows) ou .out (sous Linux).

- Exécution

Le fichier exécutable est chargé en mémoire par le système d'exploitation.
La fonction main est appelée et le code du programme est exécuté ligne par ligne.

- Cycle de vie d'une variable:

Une variable est créée lorsqu'elle est déclarée.
La variable est stockée en mémoire et une valeur lui est attribuée.
La variable peut être utilisée dans des expressions et des instructions.
La variable est détruite lorsqu'elle n'est plus utilisée.

I. Notion Langage C

Langage

2. Processus

- Débogage:
  Le débogage est le processus de recherche et de correction des erreurs dans un programme.
  Il existe plusieurs outils de débogage disponibles pour le langage C.
  Les outils de débogage permettent d'examiner le code ligne par ligne, de visualiser les valeurs des variables et de suivre l'appel des fonctions.

- Points importants à retenir:
  Le C est un langage compilé, ce qui signifie que le code source est traduit en langage machine avant d'être exécuté.
  Le C offre un contrôle précis sur la gestion de la mémoire.
  Le débogage est un élément important du processus de développement en C.

II. Structure Langage C

Langage

1. Types (entier)

- **Entier**:(int ou intiger)
  Entier court (16 bits) et peut représenter des valeurs comprises entre -32 768 et 32 767.
  Entier standard (32 bits sur la plupart des architectures) et peut représenter des valeurs comprises entre -2 147 483 648 et 2 147 483 647.
  Entier long (64 bits) et peut représenter des valeurs comprises entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807.

- **Types entiers non signés:**
  unsigned short int: Entier court non signé (16 bits) et peut représenter des valeurs comprises entre 0 et 65 535.
  unsigned int: Entier standard non signé (32 bits sur la plupart des architectures) et peut représenter des valeurs comprises entre 0 et 4 294 967 295.
  unsigned long int: Entier long non signé (64 bits) et peut représenter des valeurs comprises entre 0 et 18 446 744 073 709 551 615.
  unsigned long long int: Entier très long non signé (128 bits) et peut représenter des valeurs comprises entre 0 et 18 446 744 073 709 551 616 000 000 000 000.

II. Structure Langage C

Langage

1. Types (float)

- float: Nombre à virgule flottante simple précision (32 bits).
- double: Nombre à virgule flottante double précision (64 bits).
- long double: Nombre à virgule flottante quadruple précision (80 bits).

II. Structure Langage C

Langage

1. Types

- **Type void:**
- Indique l'absence de valeur.
- Utilisé pour les fonctions qui ne retournent aucune valeur.

- **Type char**
  Type de base pour stocker un seul caractère (8 bits).

- **Type string**
  Type de base pour stocker une chaine de caractére

II. Structure Langage C

Langage

2. Identificateur

- Un identificateur en langage C est un nom symbolique utilisé pour identifier différentes entités du langage, telles que des variables, des fonctions, des types de données, des structures, etc. Il est composé d'une séquence de caractères qui suit des règles spécifiques:

- Exemple d’identificateurs:
  Variable, Fonction ...

II. Structure Langage C

Langage

3. Constante

- Une constante en langage C est une valeur qui ne peut pas être modifiée après sa déclaration. Elle permet de stocker des valeurs fixes et immuables qui sont utilisées tout au long du programme
- Exemple
  TVA = 0,18

II. Structure Langage C

Langage C

4. Variables

- On dit qu'elle est "**variable**" car c'est une valeur qui peut changer pendant le déroulement du programme

5. syntaxes déclaration

```c
int nombreDeVies;
nombreDeVies = 5;
nombreDeVies = 4;
nombreDeVies = 3;
```

III. Opérateurs

1. Arithmétique

- Opérateurs binaires
  - Addition: +
  - Soustraction: -
  - Multiplication: *
  - Division: /
  - Modulo (reste de la division entière): %
- Opérateurs unaires:
  - Négation: - (ex: -x)
  - Incrémentation: ++ (ex: x++)
  - Décrémentation: -- (ex: x--)
- Structures de base
  - Variables et types de données (entiers, caractères, flottants...)
  - Opérateurs arithmétiques, logiques, comparaison
- Structures de contrôle :
  - sinon
  - changer
  - pendant que, fais... pendant que
  - pour

III. Opérateurs

Langage

1. Arithmétique

- Structures composées
  - Tableaux (tableau)
  - Structures (structure)
  - Pointeurs
  - Chaînes de caractères
- Fonctions
  - Prototypes
  - Passage de paramètres
  - Valeur de retour
  - Récursivité
- Gestion mémoire
  - Allocation dynamique (malloc/free)
  - Pointeurs et références
- Entrées/sorties
  - printf, scanf
  - Flux (stdin, stdout, stderr)
  - Fichiers

Chapitre 2: les structures conditionnelle :

Langage C

1. Structures conditionnelles(syntaxe d'utilisation, if else, switch)

- Structure conditionnelle optionnelle
  - condition ? expression_si_vraie : expression_si_fausse;
  - Exemple:

```c
int n = 5;
printf("%s", (n % 2 == 0) ? "pair" : "impair"); // affiche "impa
```

- **Structure conditionnelle alternatives:**
  - exemple :

```c
if(n % 2 == 0) {
  printf("pair");
} else {
  printf("impair");
}
```

switch

```c
switch (oper) {
  case '+' : printf("Addition"); break;
  case '-' : printf("Soustraction"); break;
  default : printf("Opérateur inconnu");
}
```

Chapitre 2: les structures itératives

Langage 

1. Structures itératives(syntaxe d'utilisation)

- **La boucle for**

La boucle permet d'exécuter un bloc de code un nombre déterminé de fois.

Syntaxe :

```
for (initialisation; condition; incrémentation) {
  // code
}
```

Le fonctionnement est le suivant :

- L'initialisation est exécutée une seule fois au début
- Ensuite à chaque itération :
  - La condition est réalisée, si vraie on exécute le bloc de code
  - Sinon on sort de la boucle pour
  - Après le bloc de code, l'incrémentation est exécutée
- Puis on réévalue la condition, etc.

Utilisation :

- Parcours d'un tableau
- Boucle avec nombre d'itérations connu à l'avance

Exemple

```
for (int i = 0; i < 10; i++) {
  printf("%d\n", i); // Affiche i de 0 à 9
}
```

Chapitre II: les structures itératives

Langage

1. Structures itératives(syntaxe d'utilisation)

- **La boucle while**

La boucle while permet de répéter un bloc de code tant qu'une condition reste vraie.

Syntaxe :

```
while (condition) {
  // code
}
```

Tant que la condition est vraie, le bloc de code est exécuté. Quand elle devient fausse, on sort de la boucle.

Utilisation :

- Boucle avec nombre d'itérations inconnu
- Boucle basée sur un événement

exemple

```
int i = 0;
while (i < n) { // n nombre aléatoire
  printf("%d\n", i);
  i++;
}
```

Chapitre II: les structures itératives

Langage

1. Structures itératives(syntaxe d'utilisation)

- **La boucle do while**

  On exécute le bloc une fois puis on évalue la condition. Si vraie, on répète, etc.

  Utilisation : Boucle devant être exécutée au moins une fois.

```c
do {
  // code
} while (condition);
```

exemple

```c
int i = 0;
do {
  printf("%d\n", i);
  i++;
} while (i < n);
```

Chapitre 3: les fonctions

Langage 

1. les fonctions

- Définition: Une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle est définie une seule fois et peut être appelée à partir de n'importe où dans le programme.

- Parties d'une fonction:
  - type_retour nom_fonction (liste des paramètres) { corps de la fonction }
  - type_retour : type de donnée renvoyé par la fonction (int, float, void, etc.)
  - nom_fonction : nom donné à la fonction
  - paramètres : variables définissant les données fournies à la fonction
  - corps : bloc d'instructions à exécuter
- Utilisation des fonctions :
  - Modularité du code : décomposer un programme en sous-tâches
  - Ré utilisabilité : ne coder qu'une fois des traitements récurrents.
- Types de fonctions :
  - Fonctions sans paramètres et sans retour Exemple : **void afficherMessage() { printf("Hello World!"); }**
  - Fonctions avec paramètres et sans retour Exemple:
  - **void direBonjour(char prenom[]) { printf("Bonjour %s!", prenom); }**
  - Fonctions avec paramètres et valeur de retour Exemple: **int addition(int a, int b) { return a + b; }**

Chapitre 3: les fonctions

Langage 

2. Procédures

- Définition: Une procédure est un bloc de code réutilisable, similaire à une fonction, mais qui n'est pas destiné à retourner de valeur.

- Parties d'une procédure :
  - type nom_procedure (liste des paramètres) { corps de la procédure
  - }
  - type : type de retour, toujours void pour une procédure
  - nom_procedure : nom donné à la procédure
  - paramètres : variables définissant les données fournies à la procédure
  - corps : bloc d'instructions à exécuter
- Utilisation des procédures :
  - Modularité du code
  - Réutilisabilité
  - Exécuter des tâches précises sans avoir besoin de valeur de retour
- Types de procédures :
  - Procédures sans paramètres Exemple : **void afficherBienvenue() { printf("Bienvenue sur notre site !"); }**
  - Procédures avec paramètres Exemple : **void bonjour(char prenom[]) { printf("Bonjour %s!", prenom); }**

Chapitre 3: les fonctions

Langage

3. Modules

- Il n'y a pas vraiment de notion de "module" en langage C comme on peut le trouver dans d'autres langages. Le C est un langage procédural qui favorise une approche modulaire, mais sans système de modules natif.
- on peut considérer en C :
  - Les fichiers sources (.c et .h) :
  - Permettent de séparer le code en plusieurs fichiers
  - Chaque .c compile dans un .o (objet)
  - Les .o sont linkés pour produire l'executable
- Les en-têtes (.h)
  - Fichiers d'interface contenant les prototypes de fonctions
  - Inclus dans les .c clients de ces fonctions

Chapitre 3: les fonctions

Langage

4. Paramètres ou arguments

- **Paramètres:**
  - Les paramètres sont les variables déclarées dans la définition d'une fonction pour recevoir les valeurs qui lui sont passées lors de l'appel de la fonction.
  - Ils représentent les entrées qu'une fonction peut recevoir.
  - Ils sont déclarés au niveau de la définition de la fonction.
- Arguments:
  - Les arguments sont les valeurs réelles fournies à une fonction lors de son appel.
  - Ils représentent les entrées effectivement passées à la fonction.
  - Ils sont précisés au niveau de l'appel de la fonction.

```c
// Définition de la fonction avec deux paramètres a et b
int addition(int a, int b) {
  return a + b;
}

int main() {
  // Appel de la fonction avec deux arguments 2 et 3
  int result = addition(2, 3);
}
```

Chapitre 3: les fonctions

Langage

5. Type de retour d'une fonction

- Il existe plusieurs types de retour possibles pour une fonction en langage C :
  - Pas de retour Une fonction ne peut rien retourner du tout. Dans ce cas, on utilise le mot-clé void :

```c
void maFonction() {
  // corps de la fonction
}
```

- Retour d'un scalaire Une fonction peut retourner une seule valeur scalaire de n'importe quel type : int, float, double, char etc.
  - Exemple avec un retour de type int :

```c
int maFonction() {
  return 15;
}
```

Chapitre 3: les fonctions

Langage

6. Prototype

- En C, un prototype de fonction est une déclaration de la fonction située avant sa définition qui spécifique :
  - Le type de retour
  - Le nom de la fonction
  - Les types de ses paramètres

```c
int addition(int a, int b); // prototype

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

Chapitre 3: les fonctions

Langage

7. Définition d'une fonction

- Voici les éléments essentiels pour définir une fonction en langage C :
  - Le type de retour
  - Indique le type de donnée renvoyée par la fonction. On utilise void si aucune valeur n'est retournée.
  - Le nom de la fonction
  - Le nom donné à la fonction, suivant les conventions de nommage (pas d'espaces, de caractères spéciaux etc.)
  - La liste des paramètres entre parenthèses
  - Permet de définir les variables qui passeront en entrée de la fonction lors de son appel. Chaque paramètre doit avoir un type et éventuellement un nom.
  - Le corps de la fonction entre accolades
  - Contient l'ensemble des instructions qui seront exécutées lors de l'appel de la fonction.
  - Exemple de définition d'une fonction C :

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

  int resultat;

  resultat = a + b;

  return resultat;

}
```

Chapitre 3: les fonctions

Langage

8. Appel d’une fonction

- Pour appeler une fonction en langage C, il faut suivre une syntaxe précise :
  - Écrire le nom de la fonction :
  - Il faut écrire le nom exact de la fonction tel qu'il a été défini.
  - Ajouter des parenthèses () après le nom :
  - Les parenthèses sont obligatoires pour indiquer qu'il s'agit d'un appel de fonction.
  - Passer les arguments entre les parenthèses :
  - S'il y a des paramètres dans la définition de la fonction, il faut leur passer des arguments (valeurs) correspondants.
  - Par exemple pour une fonction :

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

L'appel se fait comme ceci :

```c
int resultat;
resultat = addition(5, 3);
```

Chapitre 3: les fonctions

Langage

9. fonction itérative

- Une fonction itérative en langage C est une fonction qui utilise des structures de répétition comme les boucles for, while ou do... while pour répéter un traitement sur des données.

```c
int somme_entiers(int n) {

  int somme = 0, i;

  for (i = 1; i <= n; i++) {
    somme += i;
  }

  return somme;
}
```

Chapitre 3: les fonctions

Langage

9. fonction récursive

- Une fonction récursive en langage C est une fonction qui s'appelle elle-même. La récursion est donc une technique de programmation qui permet de résoudre certains problèmes de manière élégante, comme les algorithmes mathématiques complexes ou les parcours d'arbres et de graphiques.
    - Voici les caractéristiques d'une fonction récursive en C :
    - La fonction s'appelle elle-même directement ou indirectement
    - Il doit y avoir un cas de base, qui interrompt la récursion (condition d'arrêt)
    - Les appels récursifs se font sur des jeux de données plus petits à chaque fois
    - Permet de réduire la complexité de certains algorithmes
    - Attention à ne pas provoquer une récursion infinie (sans cas de base)
    - Exemple de fonction récursive calculant la factorielle n! en C :

```c
int factorielle(int n) {
    if (n == 0) // cas de base
        return 1;
    else
        return n * factorielle(n-1); // appel récursif
}
```

Chapitre 4: les tableaux

Langage

1. vecteur

- Définition:
  - Un vecteur en langage C est un tableau à une dimension, c'est-à-dire un tableau contenant des éléments de même type

```
vecteur[0] = 10; // première case
vecteur[3] = 5; // quatrième case
```

- Syntaxe de définition:

```
type_donnes nom_vecteur[taille];
```

Exemple

```
int tab[100]; // vecteur de 100 int
char lettres[50]; // vecteur de 50 char
double matrix[10][20]; // vecteur de 10 vecteurs de 20 double (
```

Initialisation

```
int nombres[5] = {1, 2, 3, 4, 5}; // initialization des valeurs
```

Chapitre 4: les tableaux

Langage

1. vecteur

- Remplissage:
  - Remplir un vecteur en C signifie attribuer une valeur à chacun des éléments du vecteur.
  - Cela se fait généralement :
  - Élément par élément, de manière "manuelle"
  - Avec une boucle qui parcourt toutes les cases
  - En générant des nombres aléatoires
  - En lisant des données depuis l'utilisateur

```
vecteur[0] = valeur1;
vecteur[1] = valeur2;
...
vecteur[4] = valeur5;
```

Chapitre 4: les tableaux

Langage 

1. vecteur

- Affichage:
  - Pour afficher le contenu d'un vecteur en C après qu'il a été rempli, on utilise généralement une boucle qui parcourt chaque élément du vecteur.

```c
for(i = 0; i < 5; i++) {
  printf("vecteur[%d] = %d\n", i, vecteur[i]);
}
```

exemple

```c
#include <stdio.h>

#define TAILLE 5

int main() {

  int vecteur[TAILLE];

  // Remplissage du vecteur...

  // Affichage
  printf("Elements du vecteur:\n");
  for(int i = 0; i < TAILLE; i++) {
    printf("Case %d: %d\n", i, vecteur[i]);
  }

  return 0;
}
```

Chapitre 4: les tableaux

Langage

1. vecteur

- Tri
  - C'est une technique très efficace basée sur le partitionnement. On prend un pivot, et on le place correctement dans le vecteur. On applique l'algorithme récursivement sur les sous-vecteurs créés.
  - Il existe de nombreuses autres méthodes et optimisations de tri pour les vecteurs en langage :

```c
void tri_insertion(int vecteur[], int n) {
  int i, j, temp;
  for (i = 1; i < n; i++) {
    temp = vecteur[i];
    j = i - 1;
    while(j >= 0 && temp < vecteur[j]) {
      vecteur[j + 1] = vecteur[j];
      j = j - 1;
    }
    vecteur[j + 1] = temp;
  }
}
```

Chapitre 4: les tableaux

Langage

2. Tableau a plusieurs dimensions

- Définition
  - Un tableau à plusieurs dimensions, également appelé tableau multidimensionnel ou matrice, est un tableau qui possède plus d'un indice pour accéder à ses éléments.

```
type nom_tableau[taille_1][taille_2]...[taille_N];
```

exemple

```
int tab3d[3][4][2];
```

Chapitre 4: les tableaux

Langage

2. Tableau a plusieurs dimensions

- Remplissage:

```c
#include <stdio.h>

#define LIGNES 3
#define COLONNES 4

int main() {

    int mat[LIGNES][COLONNES];
    int i, j;

    // Remplissage de la matrice
    for(i=0; i < LIGNES; i++) {
        for(j=0; j < COLONNES; j++) {
            mat[i][j] = i + j;
        }
    }

    // Affichage
    for(i=0; i < LIGNES; i++) {
        for(j=0; j < COLONNES; j++) {
            printf("%d ", mat[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Chapitre 4: les tableaux

2. Tableau a plusieurs dimensions

- Tri:

```c
// Tri par lignes
void tri_lignes(int mat[N][M]) {

  int i, j, k;

  for (i = 0; i < N; i++) {

    // tri bulles sur chaque ligne
    for (j = 0; j < M - 1; j++) {
      for (k = 0; k < M - j - 1; k++) {
        if (mat[i][k] > mat[i][k + 1])
          echange(&mat[i][k], &mat[i][k + 1]);
      }
    }
  }
}
```

Chapitre 4: les tableaux

2. Tableau a plusieurs dimensions

- **Avantage**:
  - Structure de données simple à utiliser
  - Accès rapide et efficace aux éléments
  - Permettent de représenter des données matricielles (matrices mathématiques, images, etc.)
  - Code facile à lire et à maintenir avec des index pour aux éléments
  - Occupant des zones contigües en mémoire pour une meilleure localité spatiale et temporelle
- **Inconvénients** :
  - Taille fixe définie à la compilation, difficile à redimensionner
  - Risques de dépassement de tableau si on accède à des index invalides
  - Plus le nombre de dimensions est grand, plus le code est complexe
  - Moins d'abstraction que d'autres structures de données de plus haut niveau

Chapitre 5: les enregistrements

Langage

1. Définition

- Un enregistrement est un type de données agrégé qui regroupe plusieurs variables de types différents en une seule unité. Les variables membres d'un enregistrement sont appelées champs.

- **Syntaxe**

```
struct personne p;
p.nom = "Jean";
p.age = 25;
p.taille = 1.80;
```

Exemple

```
struct personne {
  char* nom;
  int age;
  float taille;
};
```

Chapitre 5: les enregistrements

Langage

2. Syntaxe de définition

- La syntaxe de définition des enregistrements en langage C est la suivante :

```c
struct personne {
  char* nom;
  int age;
  float taille;
};
```

- **Avantages et Inconvénients**

les enregistrements sont utiles pour regrouper des données liées et simplifier l'accès aux données membres,mais ils peuvent être limités par leur taille fixe et leur sensibilité aux modifications de structure .

Chapitre 5: les enregistrements

Langage

3. Lire ou écrire une Variable de type Enregistrement:

- Pour écrire une variable de type enregistrement en langage C, vous pouvez également utiliser l'opérateur point (.). Par exemple, pour écrire le champ nom de l'enregistrement personne avec la valeur "John", vous pouvez utiliser le code suivant

```c
#include <stdio.h>

You, il y a 24 heures | 1 author (You)
struct personne
{
    char *nom;
    int age;
    float taille;
};

int main()
{
    // Créer un enregistrement personne
    struct personne personne;

    // Lire le champ nom de l'enregistrement
    char *nom = personne.nom;

    // Écrire le champ nom de l'enregistrement
    personne.nom = "John";

    // Afficher le champ nom de l'enregistrement
    printf("Nom : %s\n", personne.nom);

    return 0;
}
```

Chapitre 5: les enregistrements

4. Les Tableaux Enregistrements

- Un tableau d'enregistrements est une structure de données qui permet de stocker une collection d'enregistrements, où chaque enregistrement est une collection de champs de données.

exemple

```c
struct personne {
    char nom[50];
    char prenom[50];
    int age;
};
```

Chapitre 5: les enregistrements

5. Avantages et Inconvénients:

- **Avantages:**
  - Organisation des données
  - Accès facile aux données
  - Traitement efficace
  - Réutilisation du code
- **Inconvénients :**
  - Taille statique
  - Gestion de la mémoire
  - Accès indirect

Chapitre 5: les enregistrements

Langage

5. Lire ou écrire une variable de type Tableau Enregistrements:

```c
struct personne {
    char nom[50];
    char prenom[50];
    int age;
};

struct personne personnes[10];

// Lire le nom de la première personne
printf("%s\n", personnes[0].nom);

// Écrire dans l'âge de la deuxième personne
personnes[1].age = 30;
```