Auteur : ZEHIR Hatem
Année universitaire : 2022/2023

Informatique
Introduction à la programmation en langage C

---

1 Concepts d'un algorithme et d'un programme

1.1 Introduction à l'algorithmique

L'algorithmique est une branche de l'informatique qui concerne la conception et l'analyse des algorithmes. Un algorithme est une séquence d'instructions bien définies qui permettent de résoudre un problème donné.

Les algorithmes sont utilisés dans de nombreux domaines, tels que les mathématiques, la physique, la biologie, la finance et bien sûr l'informatique. En informatique, les algorithmes sont utilisés pour résoudre des problèmes complexes tels que la recherche, le tri et la gestion des données.

La conception d'un algorithme peut être divisée en plusieurs étapes. Tout d'abord, il est important de comprendre le problème à résoudre et de déterminer les entrées et les sorties de l'algorithme. Ensuite, il faut définir les étapes spécifiques qui seront nécessaires pour résoudre le problème.

Les algorithmes doivent être clairs, précis et efficaces. Cela signifie qu'ils doivent être compréhensibles pour d'autres personnes qui peuvent vouloir les utiliser ou les modifier, et qu'ils doivent être capables de résoudre le problème en un temps raisonnable.

Enfin, l'analyse des algorithmes est une étape importante pour déterminer leur efficacité. L'analyse permet de déterminer combien de temps et de ressources seront nécessaires pour résoudre un problème donné à l'aide de l'algorithme.

Voici un exemple simple d'un algorithme qui permet de calculer la somme de deux nombres en utilisant l'addition :

1. Demander à l'utilisateur d'entrer le premier nombre et le stocker dans une variable appelée "A"
2. Demander à l'utilisateur d'entrer le deuxième nombre et le stocker dans une variable appelée "B"
3. Ajouter les deux nombres en utilisant l'opérateur "+" et stocker le résultat dans une variable appelée "somme"
4. Afficher le résultat de la somme à l'utilisateur
5. Fin de l'algorithme

1.2 Définition et exemples de programmes

Un programme est une suite d'instructions logiques écrites dans un langage de programmation, qui permettent de réaliser une tâche spécifique sur un ordinateur.

Les programmes peuvent être très simples ou très complexes, selon la tâche qu'ils sont conçus pour réaliser. Certains programmes peuvent être utilisés pour effectuer des tâches très courantes, comme l'affichage d'un message à l'écran, tandis que d'autres peuvent être utilisés pour résoudre des problèmes complexes, tels que la simulation de phénomènes physiques.

Les programmes peuvent être écrits dans de nombreux langages de programmation différents, tels que C, Java, Python et bien d'autres. Chacun de ces langages a ses propres règles et sa propre syntaxe, mais tous les programmes écrits dans ces langages ont en commun le fait qu'ils sont traduits en langage machine pour être exécutés par l'ordinateur.

Voici quelques exemples de programmes simples :

- Un programme qui affiche "Bonjour!" à l'écran.
- Un programme qui calcule la somme de deux nombres et affiche le résultat à l'écran.
- Un programme qui demande à l'utilisateur de saisir un nombre et affiche si ce nombre est pair ou impair.
- Un programme qui affiche la table de multiplication d'un nombre donné.

Ces exemples simples illustrent comment les programmes peuvent être utilisés pour effectuer une tâche spécifique sur un ordinateur. En écrivant un programme, il est possible d'automatiser des tâches répétitives, de résoudre des problèmes complexes, ou même de créer des applications logicielles complètes.

2 Le langage C

Le langage C est un langage de programmation populaire qui a été développé dans les années 1970. Il est souvent utilisé pour écrire des programmes système, des pilotes de périphériques et des logiciels d'application.

Le langage C est un langage de programmation structuré, ce qui signifie qu'il utilise des structures de contrôle telles que les boucles et les conditions pour organiser les instructions du programme. Il utilise également une syntaxe rigoureuse pour définir les types de données, les fonctions et les variables.

3 Les éléments de base d'un programme en C

3.1 La structure de base d'un programme en C

Un programme en langage C est composé de plusieurs éléments de base, chacun ayant un rôle spécifique dans la structure globale du programme. Voici les éléments de base d'un programme en C :

- **Les directives d'inclusion de fichiers** : ce sont des instructions qui permettent d'inclure d'autres fichiers de code source dans le programme. Ces fichiers peuvent contenir des définitions de fonctions, des constantes, des macros et d'autres éléments de base du programme.
- **La fonction principale** : c'est la fonction qui est exécutée en premier lorsque le programme est lancé. Elle est généralement appelée "main()" et contient toutes les instructions qui doivent être exécutées pour réaliser la tâche du programme.
- **Les variables** : ce sont des espaces de stockage qui permettent de stocker des données dans le programme. Les variables peuvent être de différents types, tels que des entiers, des flottants, des chaînes de caractères, etc.

- **Les instructions** : ce sont des instructions qui permettent d'effectuer des opérations sur les variables. Les instructions peuvent être de différents types, tels que des affectations de valeurs, des opérations arithmétiques, des entrées/sorties, etc.

Voici un exemple simple de la structure de base d'un programme en C :

```c
#include <stdio.h> // inclusion de la directive d'inclusion de fichier

int main() // definition de la fonction principale
{
    int x = 10; // definition d'une variable entiere

    printf("La valeur de x est : %d\n", x); // affichage de la valeur de x a l'ecran

    return 0; // fin de la fonction principale
}
```

Dans cet exemple, le programme inclut le fichier d'en-tête "stdio.h", définit la fonction principale "main()", crée une variable entière "x" et affiche sa valeur à l'écran à l'aide de l'instruction "printf()". Le programme se termine ensuite avec l'instruction "return 0;".

3.2 Les fichiers d'en-tête

En langage C, les fichiers d'en-tête et les bibliothèques sont des éléments importants pour développer des programmes. Ils fournissent des définitions de fonctions, de variables et de constantes, ainsi que des prototypes et des déclarations pour les fonctions externes.

Les fichiers d'en-tête sont des fichiers de code source qui contiennent des définitions et des déclarations de fonctions, de variables et de constantes que le programme utilise. Les fichiers d'en-tête sont inclus dans le programme en utilisant la directive "#include". Les fichiers d'en-tête les plus couramment utilisés sont "stdio.h", "stdlib.h" et "math.h".

Voici un exemple de programme en C qui utilise la bibliothèque mathématique pour calculer la racine carrée d'un nombre :

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

int main()
{
    int x = 16;
    float result = sqrt(x); // appel de la fonction sqrt() de la bibliotheque mathematique

    printf("La racine carree de %f est %f\n", x, result);

    return 0;
}
```

Dans cet exemple, le programme inclut les fichiers d'en-tête "stdio.h" et "math.h", définit la fonction principale "main()", crée une variable "x" et calcule la racine carrée de "x" en utilisant la fonction "sqrt()" de la bibliothèque mathématique. Le résultat est affiché à l'écran à l'aide de l'instruction "printf()".

3.3 Instructions de lecture et d'écriture standard

En C, les fonctions "printf" et "scanf" sont utilisées pour l'entrée et la sortie standard.

La fonction "printf" est utilisée pour imprimer les valeurs sur la sortie standard, tandis que la fonction "scanf" est utilisée pour lire les entrées de l'utilisateur depuis la console.

3.3.1 La fonction printf()

La syntaxe de base de printf est la suivante :

```
printf("format de sortie", arguments);
```

Le format de sortie est une chaîne de caractères qui spécifie la manière dont les arguments doivent être imprimés. Les arguments sont des variables ou des constantes qui doivent être imprimées.

Voici un exemple :

```
int num = 5;
printf("Le nombre est %d\n", num);
```

Cet exemple imprime "Le nombre est 5" sur la sortie standard.

3.3.2 La fonction scanf()

La syntaxe de base de scanf est la suivante :

```
scanf("format d'entree", arguments);
```

Le format d'entrée est une chaîne de caractères qui spécifie la manière dont les arguments doivent être lus. Les arguments sont des adresses de variables où les entrées lues par scanf seront stockées.

Voici un exemple :

```
int num;
scanf("%d", &num);
```

Cet exemple lit un entier entré par l'utilisateur depuis la console et le stocke dans la variable num.

Il est important de noter que les formats de sortie et d'entrée doivent correspondre aux types de données des arguments respectifs pour éviter les erreurs d'exécution ou les comportements indésirables.

3.4 Les commentaires

Un commentaire en programmation est une sorte de note ou de message qu'un programmeur peut écrire pour expliquer ce que fait une partie de son code. Les commentaires ne sont pas lus ou exécutés par l'ordinateur, mais ils aident les humains à comprendre le code et à le modifier plus facilement.

Les commentaires en C sont généralement introduits par les caractères "/*" et se terminent par "*/" pour les commentaires multilignes, ou par "//" pour les commentaires sur une seule ligne. Il est recommandé d'utiliser des commentaires significatifs et de manière judicieuse pour améliorer la lisibilité du code. Il est également conseillé de maintenir les commentaires à jour lorsque des modifications sont apportées au code.

Exemple de commentaires en C :

```
/* Ce programme calcule la somme de deux nombres entiers */
#include <stdio.h>

```

```c
int main() {
    int a, b, somme;

    // Saisie des deux nombres
    printf("Entrez deux nombres entiers : ");
    scanf("%d %d", &a, &b);

    // Calcul de la somme
    somme = a + b;

    // Affichage de la somme
    printf("La somme de %d et %d est %d\n", a, b, somme);

    return 0;
}
```

Dans cet exemple, les commentaires ont été utilisés pour expliquer le but de chaque section du code, ce qui rend le programme plus facile à comprendre.

3.5 Les variables et les types de données en C

En langage C, les variables sont des éléments importants pour stocker des données dans un programme. Les variables sont déclarées en précisant leur nom et leur type. Le type de données spécifie le format de stockage de la variable et les opérations que l'on peut effectuer sur cette variable.

Les types de données primitifs en C sont :

- int : stocke les nombres entiers.
- float : stocke les nombres à virgule flottante simples précision.
- double : stocke les nombres à virgule flottante double précision.
- char : stocke les caractères.
- void : représente l'absence de type.

Il existe également des types de données définis par l'utilisateur, tels que les structures et les énumérations, qui permettent de créer des types de données personnalisés.

Voici un exemple de programme en C qui utilise des variables pour stocker des données :

```c
#include <stdio.h>

int main()
{
    int x = 10; // declaration d'une variable de type int avec une valeur de 10
    float y = 3.14; // declaration d'une variable de type float avec une valeur de 3.14
    char c = 'a'; // declaration d'une variable de type char avec la valeur 'a'

    printf("La variable x contient la valeur %d\n", x);
    printf("La variable y contient la valeur %f\n", y);
    printf("La variable c contient la valeur %c\n", c);

    return 0;
}
```

Dans cet exemple, le programme définit des variables "x", "y" et "c" avec des types et des valeurs différents. Les valeurs des variables sont affichées à l'écran à l'aide de l'instruction "printf()".

3.6 Les constantes en C

En langage C, les constantes sont des valeurs immuables qui sont utilisées dans un programme pour représenter des données qui ne doivent pas être modifiées. Les constantes peuvent être de différents types de données, tels que des entiers, des flottants ou des caractères.

Il existe deux types de constantes en C : les constantes littérales et les constantes symboliques.

Les constantes littérales sont des valeurs qui sont écrites directement dans le code source du programme. Par exemple, "10" est une constante littérale de type entier, "3.14" est une constante littérale de type flottant et "'a'" est une constante littérale de type caractère.

Les constantes symboliques sont des identificateurs qui représentent des valeurs constantes. Ils sont définis à l'aide de l'instruction "#define". Par exemple, si nous voulons définir une constante symbolique qui représente la valeur de pi, nous pouvons écrire :

```
#define PI 3.14159
```

Dans cet exemple, "PI" est un identificateur qui représente la valeur 3.14159.

Voici un exemple de programme en C qui utilise des constantes pour stocker des données :

```
#include <stdio.h>

#define PI 3.14159

int main()
{
    const int x = 10; // declaration d'une constante de type entier avec une valeur de 10
    const float y = PI; // declaration d'une constante de type float avec la valeur de la constante symbolique PI

    printf("La constante x contient la valeur %d\n", x);
    printf("La constante y contient la valeur %f\n", y);

    return 0;
}
```

Dans cet exemple, le programme définit des constantes "x" et "y" avec des types et des valeurs différents. Les valeurs des constantes sont affichées à l'écran à l'aide de l'instruction "printf()".

4 Les opérateurs en C

En langage C, les opérateurs arithmétiques sont utilisés pour effectuer des opérations mathématiques sur les variables numériques. Les opérateurs arithmétiques de base comprennent l'addition (+), la soustraction (-), la multiplication (*) et la division (/).

En voici des exemples d'utilisation :

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

```c

int main()
{
    int a = 10;
    int b = 5;

    int sum = a + b; // addition
    int difference = a - b; // soustraction
    int product = a * b; // multiplication
    int quotient = a / b; // division

    printf("La somme de a et b est %d\n", sum);
    printf("La difference de a et b est %d\n", difference);
    printf("Le produit de a et b est %d\n", product);
    printf("Le quotient de a et b est %d\n", quotient);

    return 0;
}
```

En plus des opérateurs arithmétiques de base, il existe également des opérateurs de modulos (%) et d'incrémentation (++) et de décrémentation (–). L'opérateur de modulo calcule le reste de la division entre deux nombres, tandis que les opérateurs d'incrémentation et de décrémentation sont utilisés pour augmenter ou diminuer la valeur d'une variable de 1.

Voici un exemple d'utilisation de l'opérateur de modulo :

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 3;

    int remainder = a % b; // modulo

    printf("Le reste de la division de a par b est %d\n", remainder);

    return 0;
}
```

4.1 Les opérateurs de comparaison en C

En langage C, les opérateurs de comparaison sont utilisés pour comparer des valeurs numériques ou caractères. Les résultats de ces comparaisons sont des expressions booléennes qui peuvent être soit vrai (1) ou faux (0). Les opérateurs de comparaison incluent :

- L'égalité (==) : Vérifie si deux valeurs sont égales.
- L'inégalité (! =) : Vérifie si deux valeurs sont différentes.
- L'infériorité (<) : Vérifie si une valeur est inférieure à une autre.
- La supériorité (>) : Vérifie si une valeur est supérieure à une autre.
- L'infériorité ou égalité (<=) : Vérifie si une valeur est inférieure ou égale à une autre.
- La supériorité ou égalité (>=) : Vérifie si une valeur est supérieure ou égale à une autre.

Voici un exemple d'utilisation de ces opérateurs de comparaison :

```c
#include <stdio.h>

int main()
{
    int a = 5;
    int b = 10;

    // Verifie si a est egal a b
    if (a == b)
    {
        printf("a est egal a b\n");
    }
    else
    {
        printf("a n'est pas egal a b\n");
    }

    // Verifie si a est inferieur a b
    if (a < b)
    {
        printf("a est inferieur a b\n");
    }
    else
    {
        printf("a n'est pas inferieur a b\n");
    }

    return 0;
}
```

Il est important de noter que les opérateurs de comparaison peuvent également être utilisés avec des variables de type caractère. Dans ce cas, les comparaisons sont effectuées en utilisant la valeur ASCII (norme de codage des caractères qui attribue une valeur numérique unique à chaque caractère) de chaque caractère.

5 Les Structures de contrôle conditionnel

5.1 Les structures de contrôle conditionnel if...else

Les structures de contrôle conditionnel if...else sont utilisées en C pour exécuter une instruction ou un bloc d'instructions si une condition est vraie, et une autre instruction ou bloc d'instructions si la condition est fausse. La syntaxe de base de la structure if...else est la suivante :

```c
if (condition)
{
    // Instructions a executer si la condition est vraie
}
else
{
    // Instructions a executer si la condition est fausse
}
```

La condition est une expression booléenne qui doit être évaluée comme vraie ou fausse. Si la condition est vraie, les instructions situées dans le bloc if sont exécutées. Si la condition est fausse, les instructions situées dans le bloc else sont exécutées.

Voici un exemple d'utilisation de la structure if...else :

```c
#include <stdio.h>

int main()
{
    int a = 5;

    if (a > 10)
    {
        printf("a est superieur a 10\n");
    }
    else
    {
        printf("a est inferieur ou egal a 10\n");
    }

    return 0;
}
```

Dans cet exemple, si la variable a est supérieure à 10, le programme affiche "a est supérieur à 10". Sinon, le programme affiche "a est inférieur ou égal à 10".

5.2 Les boucles while et do...while

Les boucles sont utilisées en C pour répéter des instructions plusieurs fois. Les boucles while et do...while sont des structures de contrôle de boucle en C.

La boucle while répète un bloc d'instructions tant qu'une condition est vraie. La syntaxe de base de la boucle while est la suivante :

```c
while (condition)
{
    // Instructions a repeter tant que la condition est vraie
}
```

La condition est une expression booléenne qui est évaluée à chaque tour de boucle. Si la condition est vraie, les instructions dans le bloc while sont exécutées. Si la condition est fausse, la boucle se termine et l'exécution du programme continue après le bloc while.

Voici un exemple d'utilisation de la boucle while :

```c
#include <stdio.h>

int main()
{
    int i = 0;

    while (i < 5)
    {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Dans cet exemple, la boucle while répète l'affichage de la valeur de la variable i tant que i est inférieur à 5. La variable i est incrémentée à chaque tour de boucle.

La boucle do...while est similaire à la boucle while, mais elle exécute le bloc d'instructions au moins une fois, avant de vérifier la condition de sortie. La syntaxe de base de la boucle do...while est la suivante :

```
do
{
    // Instructions a repeter au moins une fois
}
while (condition);
```

Le bloc d'instructions est exécuté une fois, puis la condition est vérifiée. Si la condition est vraie, le bloc d'instructions est répété jusqu'à ce que la condition soit fausse. Si la condition est fausse dès le départ, le bloc d'instructions est exécuté une seule fois.

Voici un exemple d'utilisation de la boucle do...while :

```
    #include <stdio.h>

int main()
{
    int i = 0;

    do
    {
        printf("%d\n", i);
        i++;
    }
    while (i < 5);

    return 0;
}
```

Dans cet exemple, le bloc d'instructions est exécuté une première fois, puis la condition est vérifiée. La boucle répète l'affichage de la valeur de la variable i tant que i est inférieur à 5. La variable i est incrémentée à chaque tour de boucle.

5.3 La boucle for

La boucle for est une autre structure de contrôle de boucle en C. Elle est souvent utilisée lorsque le nombre d'itérations à effectuer est connu à l'avance.

La syntaxe de la boucle for est la suivante :

```
for (initialisation; condition; incrementation)
{
    // Instructions a repeter tant que la condition est vraie
}
```

L'initialisation est une expression qui initialise la variable de contrôle de la boucle. La condition est une expression booléenne qui est évaluée à chaque tour de boucle. Si la condition est vraie, les instructions dans le bloc for sont exécutées. L'incrémentation est une expression qui modifie la valeur de la variable

de contrôle de la boucle à chaque tour.

Voici un exemple d'utilisation de la boucle for :

```c
#include <stdio.h>

int main()
{
    int i;

    for (i = 0; i < 5; i++)
    {
        printf("%d\n", i);
    }

    return 0;
}
```

Dans cet exemple, la boucle for répète l'affichage de la valeur de la variable i tant que i est inférieur à 5. La variable i est initialisée à 0, et est incrémentée de 1 à chaque tour de boucle.

La boucle for est également utilisée pour parcourir des tableaux et des chaînes de caractères. Voici un exemple :

```c
#include <stdio.h>

int main()
{
    int tableau[5] = {1, 2, 3, 4, 5};
    int i;

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

    return 0;
}
```

Dans cet exemple, la boucle for parcourt le tableau "tableau" et affiche chaque élément. La variable de contrôle i est utilisée comme index du tableau.

6 Tableaux et matrices

6.1 Définition et utilisation des tableaux

Un tableau en C est une structure de données qui permet de stocker une collection d'éléments de même type. Chaque élément est accessible à l'aide d'un index, qui est un entier qui spécifie la position de l'élément dans le tableau.

La syntaxe pour déclarer un tableau en C est la suivante :

```c
type nom_du_tableau[taille];
```

Pour accéder à un élément d'un tableau, nous utilisons l'index correspondant. L'indice du premier élément dans le tableau est 0, le deuxième élément est 1, et ainsi de suite jusqu'au dernier élément qui a pour

indice taille-1.

Voici un exemple d'utilisation de tableau en C :

```c
#include <stdio.h>

int main()
{
    int tableau[5] = {1, 2, 3, 4, 5};
    int i;

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

    return 0;
}
```

Dans cet exemple, nous avons déclaré un tableau nommé "tableau" avec 5 entiers initialisés à 1, 2, 3, 4 et 5. Nous avons ensuite parcouru le tableau à l'aide d'une boucle for et affiché chaque élément à l'écran.

Les tableaux en C peuvent être multidimensionnels, ce qui signifie qu'ils peuvent avoir plus d'une dimension. Par exemple, un tableau à deux dimensions peut être considéré comme une matrice, où chaque élément est accessible à l'aide de deux index, un pour la ligne et un pour la colonne.

Enfin, il est important de noter que les tableaux en C sont des structures de données statiques, ce qui signifie que leur taille est fixe et ne peut pas être modifiée pendant l'exécution du programme.

6.1.1 Saisie d'un tableau

La saisie d'un tableau en C peut être réalisée à l'aide d'instructions de lecture standard telles que scanf(). Voici un exemple de code pour saisir un tableau d'entiers en C :

```c
#include <stdio.h>

#define SIZE 10

int main() {
   int tableau[SIZE];
   int i;

   printf("Entrez les %d elements du tableau :\n", SIZE);
   for(i=0; i<SIZE; i++)
      scanf("%d", &tableau[i]);

   return 0;
}
```

Dans cet exemple, nous avons déclaré un tableau d'entiers de taille maximale "SIZE" et une variables entières "i". L'utilisateur est invité à saisir les éléments du tableau à l'aide d'une boucle "for".

6.1.2 Lecture d'un tableau

Pour lire un tableau en C, nous pouvons utiliser une boucle "for" pour parcourir tous les éléments du tableau et les afficher ou les utiliser pour effectuer des opérations. Voici un exemple de code pour lire un tableau d'entiers en C :

```c
#include <stdio.h>

#define SIZE 10

int main() {
   int tableau[SIZE];
   int i;

   printf("Entrez les %d elements du tableau :\n", SIZE);
   for(i=0; i<SIZE; i++)
      scanf("%d", &tableau[i]);

   printf("Le tableau saisi est : ");
   for(i=0; i<SIZE; i++)
      printf("%d ", tableau[i]);

   return 0;
}
```

Dans cet exemple, nous avons déclaré un tableau d'entiers de taille maximale "SIZE" et une variables entières "i". L'utilisateur est invité à saisir les éléments du tableau à l'aide d'une boucle "for". Enfin, le tableau saisi est affiché à l'aide d'une autre boucle "for".

6.1.3 Recherche d'un élément dans un tableau

Pour rechercher un élément dans un tableau en C, nous pouvons parcourir chaque élément du tableau en utilisant une boucle for ou while, puis comparer l'élément recherché à chaque élément du tableau. Voici un exemple de code pour rechercher un élément donné dans un tableau d'entiers en C :

```c
#include <stdio.h>
#define SIZE 10

int main() {
   int tableau[SIZE];
   int i, element, position = -1;

   printf("Entrez les %d elements du tableau :\n", SIZE);
   for(i=0; i<SIZE; i++)
      scanf("%d", &tableau[i]);

   printf("Entrez l'element a rechercher : ");
   scanf("%d", &element);

   for(i=0; i<SIZE; i++) {
      if(tableau[i] == element) {
         position = i;
         break;
      }
   }

   if(position == -1)
      printf("L'element %d n'est pas trouve dans le tableau.", element);
   else
      printf("L'element %d est trouve a la position %d dans le tableau.", element,
    position);

   return 0;
}
```

Dans cet exemple, nous avons déclaré un tableau d'entiers de taille maximale SIZE et trois variables entières "i", "position" et "element". L'utilisateur est invité à saisir les éléments du tableau à l'aide d'une boucle "for". L'utilisateur est également invité à entrer l'élément à rechercher. Ensuite, nous avons utilisé une boucle for pour parcourir chaque élément du tableau et comparer l'élément avec l'élément recherché. Si l'élément est trouvé dans le tableau, sa position est stockée dans la variable "position" et la boucle est arrêtée en utilisant l'instruction "break". Si l'élément n'est pas trouvé dans le tableau, la variable position reste à -1. Finalement, nous avons affiché un message indiquant si l'élément a été trouvé ou non, ainsi que sa position le cas échéant.

6.2 Les matrices

Une matrice en C est un tableau multidimensionnel, c'est-à-dire qu'il s'agit d'un tableau de tableaux. Les éléments d'une matrice sont accessibles à l'aide de deux indices, qui correspondent respectivement à la ligne et à la colonne.

La syntaxe pour déclarer une matrice en C est la suivante :

```
type nom_de_la_matrice[nombre_de_lignes][nombre_de_colonnes];
```

Voici un exemple de déclaration d'une matrice de type int de 3 lignes et 4 colonnes :

```
int matrice[3][4];
```

Pour accéder à un élément d'une matrice, nous utilisons les deux indices correspondants. Par exemple, pour accéder à l'élément dans la deuxième ligne et la troisième colonne, nous utilisons la syntaxe suivante :

```
matrice[1][2] = 42;
```

Dans cet exemple, nous avons affecté la valeur 42 à l'élément dans la deuxième ligne et la troisième colonne de la matrice.

Il est également possible d'initialiser une matrice lors de sa déclaration en utilisant la syntaxe suivante :

```
type nom_de_la_matrice[nombre_de_lignes][nombre_de_colonnes] = { {valeur1, valeur2, ...},
    {valeur1, valeur2, ...}, ... };
```

où "valeur1", "valeur2", etc. sont les valeurs initiales pour les éléments de la matrice.

Voici un exemple d'initialisation d'une matrice de type float de 2 lignes et 2 colonnes :

```
float matrice[2][2] = { {1.6, 2.4}, {3.3, 4.8} };
```

Dans cet exemple, nous avons initialisé une matrice nommée "matrice" avec les valeurs 1.6, 2.4, 3.3 et 4.8.

Enfin, il est important de noter que les matrices en C sont des tableaux multidimensionnels statiques, ce qui signifie que leur taille est fixe et ne peut pas être modifiée pendant l'exécution du programme.

6.2.1 Saisi et affichage d'une matrice

La saisie d'une matrice en langage C consiste à demander à l'utilisateur d'entrer les valeurs des éléments de la matrice et de les stocker dans un tableau à deux dimensions.

Voici un exemple de code pour la saisie d'une matrice de taille 3 x 3 en utilisant une boucle for imbriquée pour parcourir les éléments de la matrice :

```c
#include <stdio.h>

int main() {
    int i, j, mat[3][3];

    printf("Entrez les elements de la matrice :\n");

    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("Element[%d][%d] : ", i, j);
            scanf("%d", &mat[i][j]);
        }
    }

    printf("La matrice saisie est : \n");

    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
}
```

Dans cet exemple, une boucle "for" imbriquée est utilisée pour parcourir les éléments de la matrice et demander à l'utilisateur d'entrer chaque élément. Enfin, une autre boucle "for" imbriquée est utilisée pour afficher la matrice saisie à l'utilisateur.

6.2.2 Recherche d'un élément dans une matrice

La recherche d'un élément dans une matrice consiste à parcourir tous les éléments de la matrice et à vérifier si l'élément recherché est présent ou non. Cette opération peut être réalisée en utilisant des boucles imbriquées pour parcourir chaque ligne et chaque colonne de la matrice.

Voici un exemple de code pour rechercher un élément dans une matrice :

```c
#include <stdio.h>

int main() {
    int matrice[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
    int elementRecherche = 5;
    int i, j, trouve = 0;

    // Parcours de la matrice pour trouver l'element recherche
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            if (matrice[i][j] == elementRecherche) {
                trouve = 1;
                break;
            }
        }
        if (trouve) {
            break;
        }
    }
```

```
20
21      // Affichage du resultat de la recherche
22      if (trouve) {
23          printf("L'element %d a ete trouve dans la matrice a la position (%d, %d).\n",
        elementRecherche, i, j);
24      } else {
25          printf("L'element %d n'a pas ete trouve dans la matrice.\n", elementRecherche);
26      }
27
28      return 0;
29  }
```

Dans cet exemple, nous avons une matrice de 3x3 et nous cherchons l'élément 5. Nous parcourons la matrice à l'aide de deux boucles for imbriquées et nous vérifions si l'élément courant est égal à l'élément recherché. Si nous trouvons l'élément, nous sortons de la boucle et affichons la position de l'élément dans la matrice. Si nous parcourons toute la matrice sans trouver l'élément, nous affichons un message indiquant que l'élément n'a pas été trouvé.