Notion de Variable

Dans le monde de l'informatique, les variables sont bien plus que de simples étiquettes; elles sont les conteneurs dynamiques de l'information. Cette présentation vous plongera au cœur de la notion de variables, explorant leur rôle fondamental dans la construction et l'exécution des algorithmes séquentiels.

Université Abou Bakr Belkaïd - Tlemcen
Faculté des Sciences - Département de Mathématiques
abourachanez@gmail.com

Les Données : Variables et Constantes

Dans un programme, toutes les informations manipulées ne sont pas de la même nature. Certaines valeurs doivent évoluer au fil des opérations, tandis que d'autres doivent rester immuables. C'est là qu'interviennent les variables et les constantes, deux concepts fondamentaux pour gérer efficacement les données.

Variables

Les variables sont des emplacements mémoire dont la valeur peut être modifiée tout au long de l'exécution d'un programme. Elles agissent comme des étiquettes pour les données dynamiques, permettant de stocker et de récupérer des informations qui évoluent, comme un score, un compteur ou le résultat d'un calcul.

Constantes

À l'inverse des variables, les constantes représentent des valeurs fixes qui ne changent jamais après leur initialisation. Elles sont utilisées pour des données dont la nature est invariable, comme le nombre Pi, la vitesse de la lumière, ou des paramètres de configuration qui doivent rester stables.

Comprendre ces concepts est la première étape cruciale pour maîtriser la logique algorithmique et concevoir des programmes robustes et efficaces.

Les Variables :
Un Espace en Mémoire

Une variable est bien plus qu'un simple nom ; c'est un emplacement réservé dans la mémoire de l'ordinateur, doté d'une adresse unique. Cet espace est dédié au stockage d'une valeur qui peut changer au cours de l'exécution d'un programme.

Pensez-y comme une boîte étiquetée où vous pouvez ranger et récupérer des informations, essentielles pour toute opération algorithmique. Chaque variable a un nom, une valeur et un type.

Schéma du fonctionnement de la mémoire vive

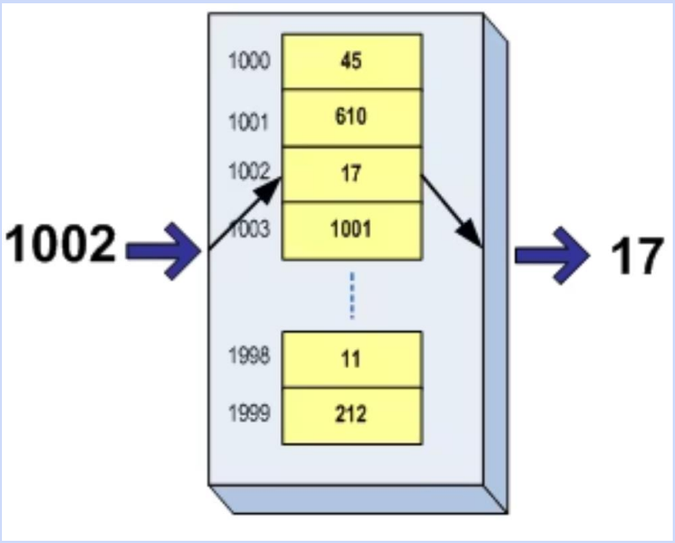
1. **Les adresses :** une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive, par exemple 1002 .

2. **Les valeurs :** à chaque adresse, on peut stocker une valeur (un nombre). On ne peut stocker qu'un nombre par adresse !

En langage C, une variable est constituée de deux choses :

1. **Un nom :** c'est ce qui permet de la reconnaître. En programmation, on n'aura pas à retenir l'adresse mémoire, on va juste indiquer des noms de variables. C'est le compilateur qui fera la conversion entre le nom et l'adresse.

2. **Une valeur :** c'est le nombre qu'elle stocke, par exemple 17 ;



Les Types de Données Fondamentaux

Chaque variable que nous déclarons doit être d'un type spécifique. Ce type définit non seulement la nature des données que la variable peut contenir, mais aussi l'espace qu'elle occupe en mémoire et les opérations que l'on peut effectuer sur elle. Comprendre ces types fondamentaux est essentiel pour écrire des programmes corrects et efficaces.

Entiers (int, short, long)

Utilisés pour stocker des nombres **entiers**, sans partie décimale. Ils sont idéaux pour les quantités, les compteurs ou les indices.

Plages de valeurs :

- **short** (entier court) : –32 768 à +32 767
- **int** (entier standard) : –2 147 483 648 à +2 147 483 647
- **long** (entier long) : –9 223 372 036 854 775 808 à +9 223 372 036 854 775 807

☞ En pratique, on utilise surtout **int** (suffisant pour la majorité des calculs), et **long** quand on manipule des nombres très grands.

Nombres Flottants (float, double)

Ces types servent à représenter des nombres avec décimales.

Plages de valeurs :

- **float** : précision simple, environ 7 chiffres significatifs : ±3.4e-38 à ±3.4e+38
- **double** : double précision, environ 15 chiffres significatifs : ±1.7e-308 à ±1.7e+308

☞ En pratique, on utilise généralement **double** pour les calculs scientifiques car il est plus précis.

Caractères (char)

Permettent de stocker un seul caractère : une lettre, un chiffre, un symbole spécial. En C, les caractères sont traités comme de petits nombres entiers correspondant à leur code ASCII.

Plage de valeurs : **char** : Généralement -128 à 127 ou 0 à 255 (selon si signé ou non signé).

Booléens (bool)

Représentent des valeurs logiques : VRAI (true) ou FAUX (false). Ils sont fondamentaux pour les structures de contrôle (conditions, boucles) et pour gérer les états binaires dans un programme.

Plage de valeurs : **bool**: true ou false.

Nommer une variables

Dans la plupart des langages de programmation, dont le C, il existe des **contraintes** et **conventions** à respecter :

- Une variable peut contenir uniquement : **lettres (minuscules/majuscules), chiffres et underscore _**.
- Le **premier caractère** doit être une lettre (jamais un chiffre).
- Les **espaces sont interdits** → utilisez _ à la place.
- Pas d'**accents** ni de caractères spéciaux (é , â , - , ? , # , etc.).
- Les **mots réservés** du langage (`int`, `for`, `return`, …) ne peuvent pas être utilisés comme noms de variables.

Exemple : Quels sont les noms de variables corrects ?

noteAlgo ✓	mail ✓	Prix-TTC ✗	nom ✓
numero_de_telephone ✓	bac2025 ✓	main ✗	prénom ✗

Bonnes pratiques pour nommer les variables en C

La clarté et la cohérence dans la nomination des variables sont fondamentales pour la lisibilité, la compréhension et la maintenabilité de votre code. De bonnes habitudes dès le départ simplifient grandement le travail d'équipe et la relecture future.

Noms clairs et significatifs

Choisissez des noms qui décrivent précisément l'objet ou l'objectif de la variable (ex: `nombreClients`, `temperatureMaximale`). Évitez les noms trop courts ou ambigus comme `x` ou `temp`, sauf pour les boucles très courtes.

Conventions de nommage

Adoptez une convention de nommage cohérente dans un programme:

☞ En **C**, on utilise souvent :

- **snake_case** → exemple : `nombre_clients`
- **camelCase** → exemple : `nombreClients`
- Les **constantes** sont généralement écrites en **MAJUSCULES** avec des underscores → exemple : `MAX_VALEUR`

Exemple comparatif

```c
// Mauvais nommage
int x, y;
float var1;
int NombreTOTALDesEtudiants;
```

```c
// Bon nommage
int age_etudiant;
float moyenneClasse;
int nombre_etudiants;
const float PI = 3.14159;
```

Déclarer une variable

Avant d'utiliser une variable en C, il est impératif de la déclarer. La déclaration est l'acte d'informer le compilateur du **nom** de la variable et du **type de données** qu'elle va contenir. Cette étape fondamentale permet au compilateur de réserver l'espace mémoire nécessaire et de s'assurer que les opérations effectuées sur cette variable sont cohérentes avec son type.

```
int age;                    // Déclaration d'une variable entière 'age‘
double prix = 19.99;        // Déclaration et initialisation d'une variable flottante 'prix‘
char initiale = 'A';        // Déclaration et initialisation d'une variable caractère 'initiale'
```

La syntaxe de déclaration est directe :

Vous spécifiez d'abord le type de données (comme int ou double), suivi du nom que vous choisissez pour la variable. Il est également possible, et souvent recommandé, d'initialiser la variable avec une valeur dès sa déclaration, comme illustré avec prix et initiale. Cela permet de s’assurer que la variable contient une valeur correcte dès le départ, et non une valeur indéfinie (du "n’importe quoi").

Affecter une valeur à une variable

Après avoir déclaré une variable, la prochaine étape essentielle est de lui attribuer une **valeur.** Cette opération, appelée affectation, utilise l'opérateur d'affectation (=) pour stocker une donnée spécifique dans l'emplacement mémoire réservé par la variable. C'est ainsi que la variable devient utile et peut participer aux calculs et logiques de votre programme.

```
*  int compteur;                  // Déclaration
   compteur = 10;                 // Affectation: 'compteur' prend la valeur 10

*  double resultat;               // Déclaration
    resultat = 5.0 / 2.0;         // Affectation: 'resultat' prend la valeur 2.5

*  int age = 1;                   // Déclaration et initialisation (affectation à la déclaration)
   age = 18;                     // Nouvelle affectation: ‘age' prend la valeur 18
```

👉 Dans cet exemple, la variable `age` montre qu’il est possible d’initialiser une variable dès sa déclaration, puis de modifier sa valeur par la suite.

Affichage des variables avec `printf()`

La fonction `printf()` est l'outil standard en C pour afficher du texte et des valeurs de variables sur la console. Elle permet un affichage formaté, rendant la sortie de votre programme claire et lisible. C'est essentiel pour interagir avec l'utilisateur.

On utilise `printf()` de la même manière que pour afficher un texte, sauf que l'on rajoute un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable.

La fonction `printf()` peut afficher une ou plusieurs variables dans une seule instruction, en plaçant les spécificateurs de format dans la chaîne de caractères et les variables correspondantes après la virgule, dans le même ordre.

Les spécificateurs de format pour `printf()`

Ces caractères spéciaux indiquent à `printf()` le type de données de la variable à afficher. Il est crucial de faire correspondre le spécificateur au type de la variable pour éviter des erreurs ou un affichage incorrect.

%d

Pour un nombre **entier** (`int`, `short`, `long`).

%f

Pour un nombre **réel** (`float` ou `double`).
On peut contrôler le nombre de chiffres après la virgule en précisant la précision, par exemple `%.2f` pour 2 décimales.

%c

Pour un **caractère unique** (`char`).

%s

Pour une **chaîne de caractères** (tableau de `char`).

Exemples d'affichage

```
# include <stdio.h>              // Nécessaire pour printf()
# include <stdlib.h>
int main()
 {
    int age = 20;
   double taille = 1.72;

        // Afficher une seule variable
   printf("Mon age est : %d ans.\n", age);                  // Le caractère \n provoque un retour à la ligne
      // Résultat : Mon age est : 20 ans.

      // Afficher plusieurs variables
   printf("J'ai %d ans et mesure %.2f m.\n", age , taille);
     // Résultat : J'ai 20 ans et mesure 1.72 m.
   return 0;  }
```

Bonnes pratiques

- **Correspondance des types :** Assurez-vous que le spécificateur de format correspond toujours au type de la variable.
- **Ordre des arguments :** L'ordre des spécificateurs doit correspondre à l'ordre des variables après la chaîne de format.
- **Clarté :** Utilisez des messages descriptifs pour que la sortie soit facile à comprendre.
- **Précision :** Pour les flottants, spécifiez la précision (ex: `%.2f`) pour un affichage propre.

Saisie des variables avec `scanf()`

Pour rendre vos programmes C interactifs, vous aurez besoin de moyens pour lire les informations fournies par l'utilisateur via le clavier. La fonction standard la plus courante pour cela est `scanf()`.

La fonction `scanf()` complète `printf()` : elle sert à lire des valeurs, tandis que `printf()` sert à les afficher.

Les spécificateurs de format pour `scanf()`

Les spécificateurs de format utilisés avec `scanf()` sont les mêmes que pour `printf()`, à l'exception du type double qui utilise `%lf` au lieu de `%f`.

👉 On doit mettre le spécificateurs de format entre guillemets, par exemple: "%d", puis indiquer le nom de la variable qui va recevoir la valeur. Par ailleurs, il faut mettre le symbole & devant le nom de la variable.

👉 L'opérateur d'adresse & signifie "l'adresse mémoire de" et indique à `scanf()` où elle doit écrire la valeur lue.

Exemples de saisie

```
# include <stdio.h>

int main()
 {
    int age;      double taille;

    printf("Quel est votre âge ? ");
    scanf("%d", &age);          // Notez le & devant 'age'

    printf("Quelle est votre taille (en mètres) ? ");
    scanf("%lf ", &taille);     // Notez le & devant 'taille'

    printf("Donc, vous avez %d ans et mesurez %.2f m.\n", age, taille);

 return 0;
 }
```

Bonnes pratiques et précautions

- **Correspondance des types :** Toujours faire correspondre le spécificateur de format au type de la variable cible pour éviter les erreurs et les comportements imprévus.
- **Opérateur `&` :** N'oubliez jamais l'opérateur `&` pour les variables simples (`int`, `float`, `char`, etc.). Sans lui, le programme risque de planter ou de donner des résultats incorrects.

Les Variables en Action : L'Addition et la Mémoire

Lorsqu'un programme calcule la somme de deux entiers, les variables servent de conteneurs dans la mémoire vive (RAM). Chaque variable déclarée se voit attribuer une adresse mémoire unique où sa valeur sera stockée. L'ordinateur peut ensuite accéder à ces adresses pour lire ou modifier les données, ce qui est fondamental pour les opérations.

```
int main()
{
  int nombre1, nombre2, somme=0;

  printf (" Entrez le nombre 1 : ");
  scanf ("%d", &nombre1 );
  printf (" Entrez le nombre 2 : ");
  scanf ("%d", &nombre2 );

  somme = nombre1+nombre2;
     // L'ordinateur lit les valeurs, effectue l'addition, et stocke le résultat dans somme.

  printf ("%d + %d = %d \n", nombre1, nombre2, somme);          // Et on affiche l'addition à l'écran.

return 0;
}
```

Visualisation de l'Addition en Mémoire

Pour mieux comprendre comment les variables interagissent avec la mémoire lors d'une opération simple comme l'addition, examinons une représentation schématique. Chaque variable est une boîte dans la mémoire vive, identifiée par une adresse unique, et contenant une valeur.

Dans notre exemple, `nombre1` et `nombre2` résident à des adresses distinctes (0xAF01 et 0xAF02) et stockent leurs valeurs initiales. Lorsque l'addition est effectuée, le processeur accède à ces valeurs, calcule leur somme, puis stocke le résultat (15) dans l'emplacement mémoire alloué à la variable `somme` (0xAF03).