PART 2 : VARIABLES SIMPLES

- **Qu’est ce qu’une variable?**

- **Variables en C:**
  - Les types élémentaires en C
  - Déclarer des variables dans un programme
  - Contraintes pour le choix de noms

- **Manipulation de base sur les variable:**
  - Affecter une valeur à une variable
  - Les valeurs de type caractère (codage ASCII)
  - Utilisation de printf() pour afficher des valeurs
  - Obtenir et afficher la taille en mémoire d’une variable
  - Obtenir et afficher l’adresse mémoire d’une variable
  - La fonction scanf() pour récupérer une valeur entrée par l’utilisateur

Qu’est ce qu’une variable?

- Une variable est un espace mémoire réservé est accessible qui permet de stocker et d’utiliser des valeurs qui sont utilisées pendant l'exécution du programme.

- Les noms des variables sont des identificateurs quelconques.

- Il y a deux actions possibles:
  - Donner une valeur à la variable.
  - Utiliser la valeur de la variable

- Quelque soit le langage:
  - il y a toujours plusieurs type de variable.
  - Chaque type est identifié par un mot clé.

Les types en C

- Au niveau du processeur, toutes les données sont représentées sous leur forme binaire et la notion de type n'a pas de sens.

- Cette notion n'a été introduite que par les langages de haut niveau dans le but de rendre les programmes plus rationnels et structurés.

- Le langage C dispose des types de base suivants:

Types	Description
int	**entier**
float	**Réel simple précision**
double	**Réel double précision**
char	**caractère**

Les types en C - Taille des types

Type	Dimension (octets)
bool	1
char, unsigned char, signed char	1
short, unsigned short	2
int, unsigned int	4
long, unsigned long	4
float	4
double	8
Long double	8

Les types en C – Plage de valeurs

Type	*Taille (Bits)*	*Plage de valeur*	
bit	1	0 , 1	*(entier)*
char	8	-128 to 127	*(entier)*
unsigned char	8	0 to 255	*(entier)*
signed char	8	-128 to 127	*(entier)*
int	16	-32768 to 32767	*(entier)*
short int	16	-32768 to 32767	*(entier)*
unsigned int	16	0 to 65535	*(entier)*
signed int	16	-32768 to 32767	*(entier)*
long int	32	-2147483648 to 2147483647	*(entier)*
unsigned long int	32	0 to 4294967295	*(entier)*
signed long int	32	-2147483648 to 2147483647	*(entier)*
float	32	±1.175e-38 to ±3.402e38	*(réel)*
double	32	±1.175e-38 to ±3.402e38	*(réel)*

Les types en C - Les identificateurs

- Un identificateur peut être formé d'un nombre illimité de caractères alphanumériques ainsi que du caractère " underscore (_) ".

- Le premier caractère est soit une lettre ou (_). Il est déconseillé d'utiliser le caractère (_) car il peut créer des interférences avec les librairies.

- Les lettres en minuscules et en majuscules sont différentes. Un identificateur var est différent d'un identificateur Var

- Un identificateur doit être différents des mots clés utilisés par le C:

  asm, auto, break, case, char, class, const, continue, default, delete, do, double, else, enum, extern, float, for, friend, goto, if, inline, int, long, new, operator, overload, private, protected, public, register, return, short, sizeof, static, struct, switch, this, typedef, union, unsigned, virtual, void, while.

Les types en C - Les identificateurs

- **Identificateurs valides :**
  - xx y1 somme_5 _position
  - Noms surface fin_de_fichier VECTEUR
- **Identificateurs invalides :**
  - 3eme commence par un chiffre
  - x#y caractère non autorisé (#)
  - no-commande caractère non autorisé (-)
  - taux change caractère non autorisé (espace)

Les opérateurs

Affectation :

- variable = expression
- expression est évalué et est affectée à variable.
- L'affectation est une conversion de type implicite : la valeur de l'expression est convertit dans le type du terme gauche.

```
main()
{
        int i, j = 2;
        float x = 2.5;
        i = j + x;
        x = x + i;
        printf("\n %f \n",x)
}
```

Les opérations arithmétiques

- **Opérateur unaire – et opérateurs binaires : +, -, *, /, % (reste de la division = modulo)**

- **La division entière est désignée par /. Si les 2 opérandes sont de types entier, / produira une division entière. Par exemple :**
  - float x;
  - x = 3 / 2; affecte à x la valeur 1.
  - x = 3 / 2.; affect à x la valeur 1.5

- **pow(x,y) : fonction de la librairie math.h pour calculer x^y**

Les opérateurs relationnels

- **La syntaxe est expression1 op expression2:**
  - `>` strictement supérieur,
  - `>=` supérieur ou égal,
  - `<` strictement inférieur,
  - `<=` inférieur ou égal,
  - `==` égal,
  - `!=` différent

- **La valeur booléenne rendu est de type int :**
  - 1 pour vraie
  - 0 sinon

Les opérateurs logiques

- **Booléens :**
  - && : et logique
  - || : ou logique
  - ! : négation logique

Opérateurs d'affectation composée

- **+= -= *= /= &=^= |= <<= <<=**

- **Pour tout opérateur op :**

  - ***expression1 op= expression2***
    équivaut à
  - ***expression1 = expression1 op expression2***

- **expression1 n'est évalué qu'une seule fois**

Opérateurs incrémentation ou décrementation

- **Incrémentation : ++**
- **Décrémentation : --**
- **En suffixe i++ ou en préfixe ++i :** dans les deux cas la valeur de i sera incrémentée, sauf pour le premier cas la valeur affectée est l'ancienne, exemple :

```
int a = 3, b, c;
b = ++a  /* a et b valent 4*/
c = b++          /* c vaut 4 et b vaut 5 */
```

Opérateur virgule

- **Suite d'expressions séparées par une virgule**
  - expression1 , expression2 , expression 3

- **Expression évaluée de gauche à droite, sa valeur sera la valeur de l'expression de droite:**

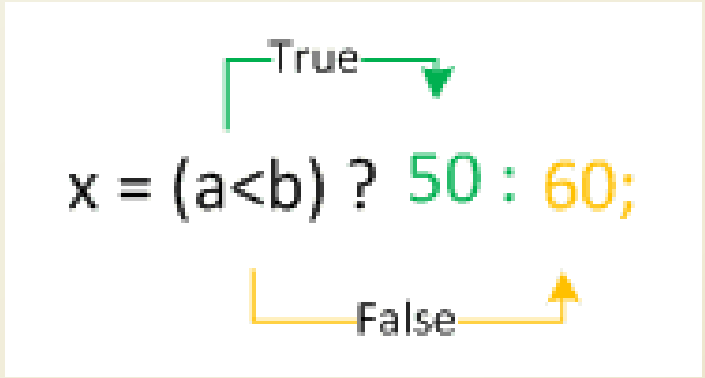
```
main()
{
    int a , b
    b = ((a = 3) , (a + 2));
    printf("\n b= %d\n",b);
}
imprime b = 5
```

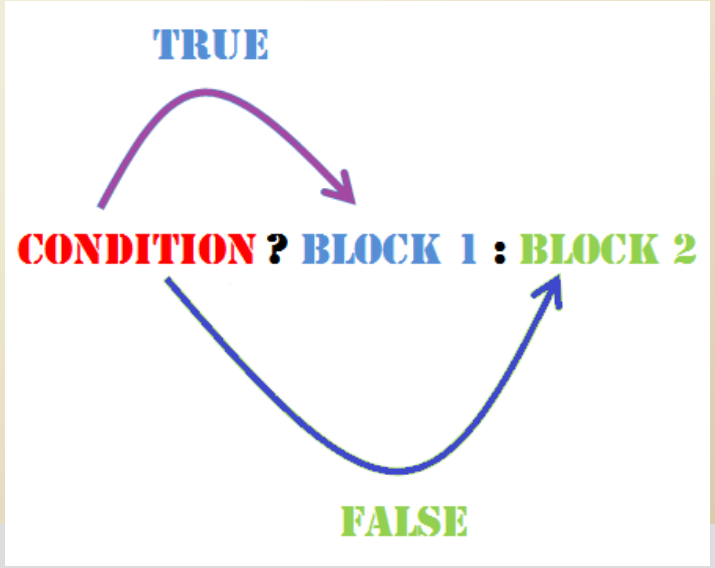
Opérateur conditionnel ternaire

- **Opérateur conditionnel : ?**
  - condition ? expression1 : expression2

- **Cette expression est égale à expression1 si la condition est vraie et à expression2 sinon**

  - x>=0 ? x : -x; correspond à la valeur absolue
  - m = ((a>b) ? a : b); affecte à m le maximum de a et b





Opérateur de conversion de type

- **Opérateur de conversion de type, appelé cast, permet de modifier explicitement le type d'un objet :**

  - **(type)** objet

```
main()
{
    int i = 3 , j = 2;
    printf("%f \n", (float) i/j);
}
```

Renvoie la valeur 1.5

Opérateur adresse

- L'opérateur d'adresse **&** appliqué à une variable retourne l'adresse-mémoire de cette valeur

&objet