



C Programming language

introduction

La langue C est un langage de programmation puissant et influent, développé dans les années 1970. Il est apprécié pour sa simplicité, son efficacité et sa portabilité. C est un langage procédural qui permet de développer des applications de bas niveau et de contrôler directement la mémoire grâce à l'utilisation de pointeurs. Il possède une syntaxe concise et un ensemble de fonctionnalités de base, en faisant une base importante pour de nombreux autres langages de programmation.





Pourquoi language C?

Les programmeurs recommandent d'apprendre C en premier car cela offre une base solide pour comprendre les concepts fondamentaux de la programmation. C est un langage simple et puissant qui permet de maîtriser les principes de base tels que les variables, les boucles, les fonctions et la gestion de la mémoire. De plus, la connaissance de C facilite ensuite l'apprentissage d'autres langages de programmation. En commençant par C, les programmeurs développent une compréhension approfondie de la logique de programmation et acquièrent des compétences transférables à d'autres domaines du développement logiciel.

Structure of C Program

<i>Header</i>	#include <stdio.h>
<i>main()</i>	int main()
<i>Variable declaration</i>	int a = 10;
<i>Body</i>	printf("%d ", a);
<i>Return</i>	}

semesters.in

Objectives

1

LES VARIABLES

2

LES OPERATIONS
DE MATH

3

CONDITIONS

4

LOOPS

LES VARIABLES

Le langage C offre plusieurs types de variables qui peuvent être utilisés pour stocker différents types de données. Voici les types de variables les plus couramment utilisés en C :

1. Entiers (Integers) :

- "int" : Utilisé pour stocker des nombres entiers, tels que 1, -5, 100, etc.
- "short" : Un entier court, utilisé pour stocker des nombres entiers plus petits que ceux stockés dans un int.
- "long" : Un entier long, utilisé pour stocker des nombres entiers plus grands que ceux stockés dans un int.
- "unsigned" : Utilisé pour stocker des nombres entiers positifs uniquement (sans signe).

2. Nombres à virgule flottante (Floating-Point) :

- "float" : Utilisé pour stocker des nombres décimaux à virgule flottante, tels que 3.14, -0.5, etc.
- "double" : Utilisé pour stocker des nombres décimaux avec une précision plus élevée que le "float".

3. Caractères (Characters) :

- "char" : Utilisé pour stocker des caractères individuels, tels que 'A', 'b', '\$', etc.

4. Chaînes de caractères (Strings) :

- "char[]" ou "char*" : Utilisé pour stocker des séquences de caractères, formant une chaîne de texte.

5. Booléens :

- "bool" : Utilisé pour stocker des valeurs de vérité, soit "true" (vrai) ou "false" (faux). Pour utiliser le type booléen, il est souvent nécessaire d'inclure la bibliothèque "stdbool.h".

VARIABLE DECLARATION/

Comment déclarer une variable

```
1 #include <stdio.h>
2 int main () {
3     int a,b,c;
4     a = 10;
5     b = 20;
6     c = a+b;
7     printf("value of c : %d", c);
8     return 0;
9 }
```

LES OPERATIONS DE MATH EN C

Les opérations mathématiques en langage C se réfèrent aux calculs fondamentaux effectués au sein d'un programme C pour manipuler des données numériques. Ces opérations comprennent l'addition, la soustraction, la multiplication, la division et le modulo (reste de la division). Les programmeurs utilisent des opérateurs spécifiques, tels que + pour l'addition, - pour la soustraction, * pour la multiplication, / pour la division et % pour le modulo, pour effectuer ces calculs. Ces opérations sont essentielles pour résoudre des problèmes mathématiques, traiter des données et mettre en œuvre divers algorithmes en programmation C.

LES TYPES D'OPERATIONS DE MATH

1. Addition (+):

- Effectue l'addition entre deux valeurs.
- Exemple : `resultat = num1 + num2;`

2. Soustraction (-):

- Effectue la soustraction entre deux valeurs.
- Exemple : `resultat = num1 - num2;`

3. Multiplication (*):

- Effectue la multiplication entre deux valeurs.
- Exemple : `resultat = num1 * num2;`

4. Division (/):

- Effectue la division entre deux valeurs.
- Exemple : `resultat = num1 / num2;`
- Remarque : Soyez prudent lors de la division par zéro, car cela peut entraîner une erreur d'exécution.

5. Modulo (%):

- Calcule le reste d'une opération de division.
- Exemple : `reste = num1 % num2;`
- Utile pour des tâches telles que vérifier si un nombre est pair ou impair.

What's conditions

Conditions in programming are like decision-making tools that help your computer programs respond to different situations. Think of them as the choices you make every day: when you see a green traffic light, you go (like an "if" condition); when it's red, you stop or do something else (like "else"). Sometimes you have multiple options (like "else if"), and other times you choose from a menu (like "switch"). These conditions help your program decide what to do based on specific conditions or values, making your code smart and adaptable.



TYPE OF CONDITIONS

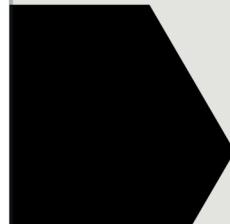
There is 4 essencial conditions types:

1. **if Statement:** The foundational conditional statement that allows you to execute code if a specific condition is true.
2. **else Statement:** Used in conjunction with **if**, it provides an alternative code block to execute when the initial condition is false.
3. **else if Statement:** Enables you to check multiple conditions sequentially, choosing the first one that is true or taking a default action if none match.
4. **Switch Statement:** Efficiently selects a single option from multiple possibilities based on the value of an expression, streamlining decision-making in code.

Example of using ALL 4 condition type

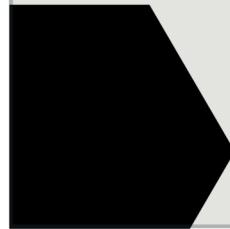
IF STATEMENT:

```
int age = 18;  
if (age >= 18) {  
    printf("You can apply to YouCode\n");  
}
```



ELSE STATEMENT

```
int temperature = 38;  
if (temperature > 45) {  
printf("Ahya m3elem t9hert b sehd\n"); }  
else {  
printf("Kayn chewia tel gherbi\n"); }
```



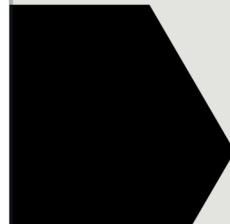
ELSE IF STATEMENT:

```
int number=12;
if (number > 0) {
printf("The number is positive.\n");
}
else if (number < 0) {
printf("The number is negative.\n");
}
else {
printf("The number is zero.\n");
}
```



SWITCH STATEMENT:

```
int day;
printf("Enter a number (1-7): ");
scanf("%d", &day);
switch (day) {
    case 1: printf("Tnina\n");
    break;
    case 2: printf("Tlat\n");
    break;
    case 3: printf("Larbe3\n");
    break;
    case 4: printf("Khemiss\n");
    break;
    case 5: printf("Jem3a\n");
    break;
    case 6: printf("Sebt\n");
    break;
    case 7: printf("IHed\n");
    break;
default: printf("Invalid day number. Please enter a number between 1 and 7.\n"); }
```



LOOPS



```
state={  
  products: storeProducts  
}  
render(){  
  return (  
    <React.Fragment>  
      <div className="py-5">  
        <div className="container">  
          <Title name="our" title="Our Products" />  
          <div className="row">  
            <ProductConsumer>  
              {(value) => {  
                console.log(value);  
              }}  
            </ProductConsumer>  
          </div>  
        </div>  
      </React.Fragment>  
    )  
  );  
}
```

What's loops

Loops in C are like magical repeat buttons for your computer program. Imagine you have a task, like counting from 1 to 10. Instead of writing each number one by one, you can use a loop to say, "Hey computer, do this for me again and again until I say stop!" The computer listens and goes, "Sure, I'll count for you." That's what loops do - they help you repeat a task as many times as you want, making your programs efficient and saving you from doing the same thing over and over by hand. It's like having a little helper that follows your instructions to the letter.



TYPE OF LOOPS

There is 3 essencial loops types:

1. **for Loop:** Think of it like counting on your fingers. You start at a number, keep counting up or down, and stop when you reach a certain point. The for loop is great when you know exactly how many times you want to do something.
2. **while Loop:** Imagine you're playing a game and you want to keep playing as long as you have energy left. You check if you have energy before each round. If you do, you keep playing; otherwise, you stop. The while loop is similar – it keeps going as long as a condition is true.
3. **do-while Loop:** Picture a situation where you want to play a game at least once, and then you ask if you want to play again. You play once, ask if you want to continue, and if you say "yes," you play another round. The do-while loop works like that – it ensures you do something at least once and then checks if you want to do it again.

Example of using ALL 3 loops type

FOR LOOP:

```
for (int i = 1; i <= 5; i++)  
{  
    printf("Count: %d\n", i);  
}
```

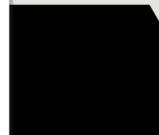
In this for loop, we start at 1, count up to 5, and print the count at each step.



WHILE LOOP:

```
int energy = 3;  
while (energy > 0)  
{   Ajouter des lignes dans le corps du texte  
    printf("Playing the game!\n");  
    energy--;  
}
```

Here, the while loop keeps playing the game as long as you have energy (greater than 0).



FOR LOOP:

```
int playAgain;
do {
    printf("Playing the game!\n");
    printf("Do you want to play again? (1
for Yes, 0 for No): ");
    scanf("%d", &playAgain);
} while (playAgain == 1);
```

In this do-while loop, you play the game at least once, and then it asks if you want to play again. If you say "Yes" (1), it plays another round; otherwise, it stops.



**Thank
You!**