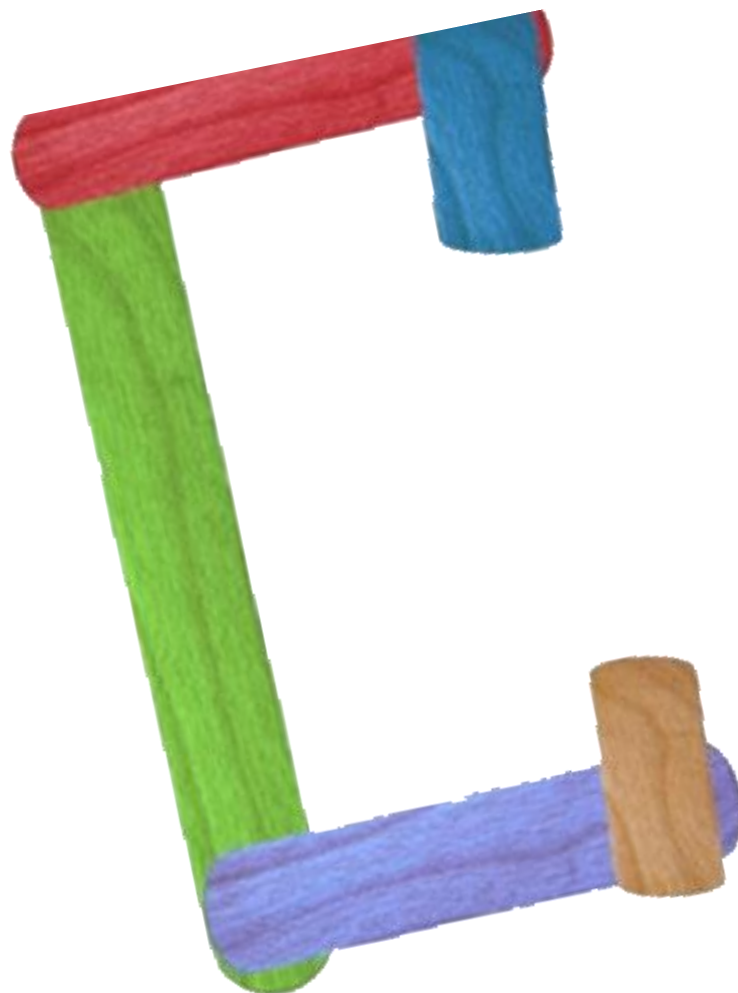
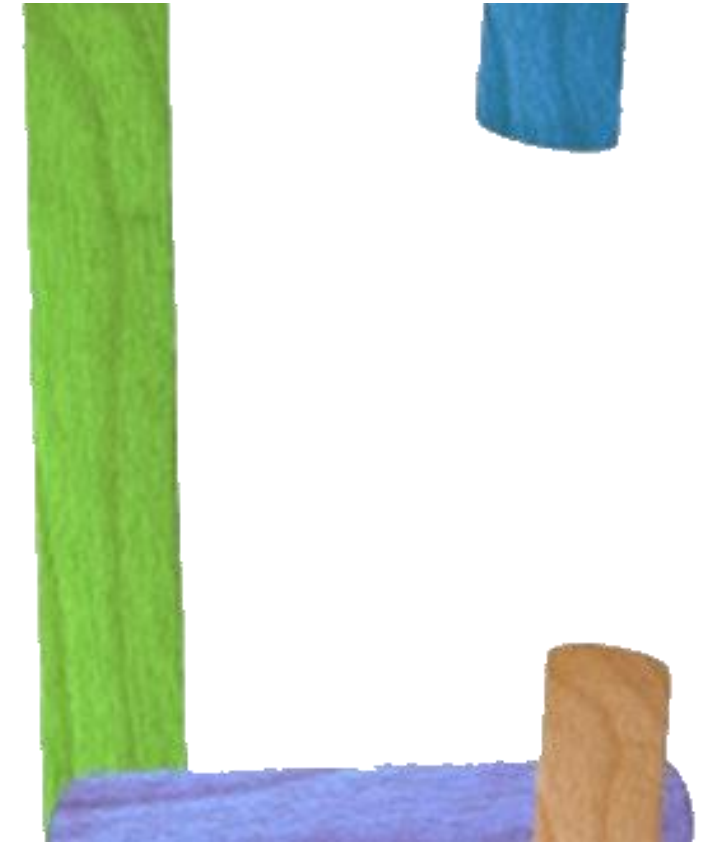


Linguagem



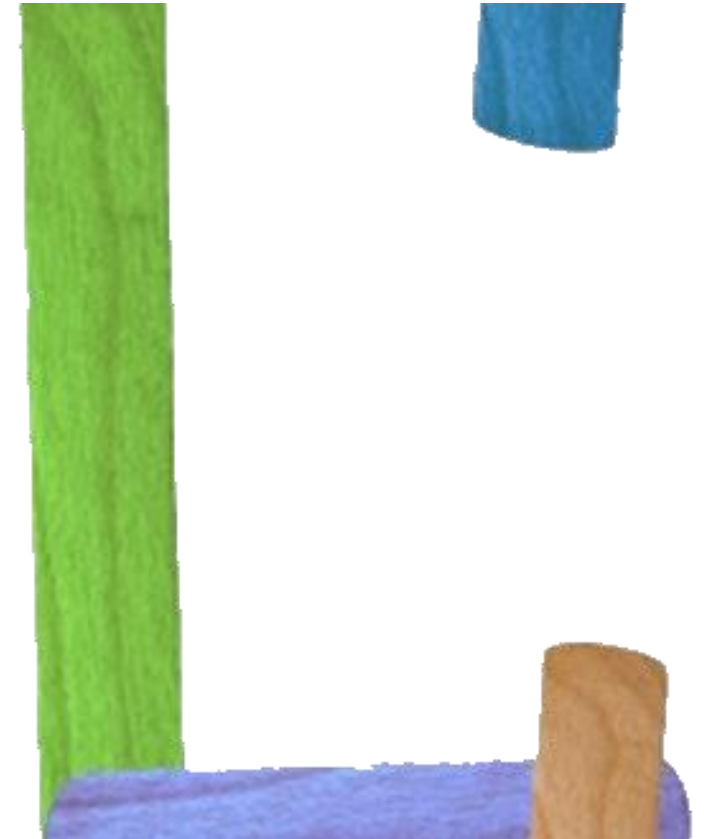
Devido às suas características e eficiência, a linguagem C é amplamente utilizada no desenvolvimento de sistemas operacionais, compiladores, aplicativos de alto desempenho, jogos, dispositivos embarcados e muitos outros domínios de desenvolvimento de software



A linguagem C é uma linguagem de programação de propósito geral que foi originalmente desenvolvida por Dennis Ritchie nos anos 1970 no Bell Labs.

Simplicidade: A sintaxe da linguagem C é relativamente simples e concisa, facilitando a leitura e escrita de código. A linguagem possui um conjunto pequeno de palavras-chave e regras de estruturação claras.

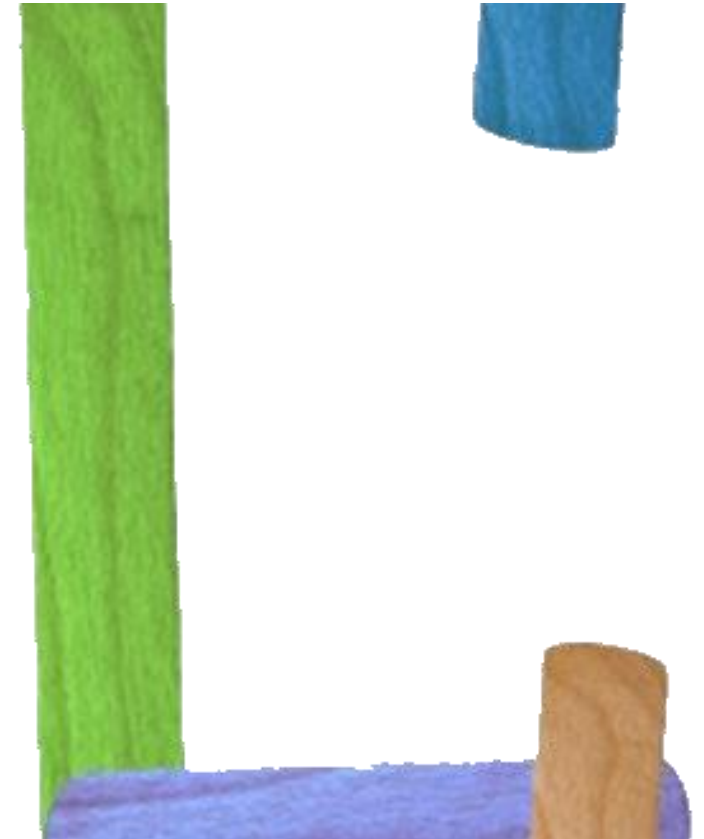
Eficiência: C é conhecida por sua eficiência em termos de uso de memória e desempenho. Ela permite acesso direto à memória e possui um sistema de tipos que permite manipular dados de forma eficiente.



A linguagem C é uma linguagem de programação de propósito geral que foi originalmente desenvolvida por Dennis Ritchie nos anos 1970 no Bell Labs.

Portabilidade: C é uma linguagem de programação portátil, o que significa que os programas escritos em C podem ser compilados e executados em diferentes plataformas e sistemas operacionais com poucas ou nenhuma modificação.

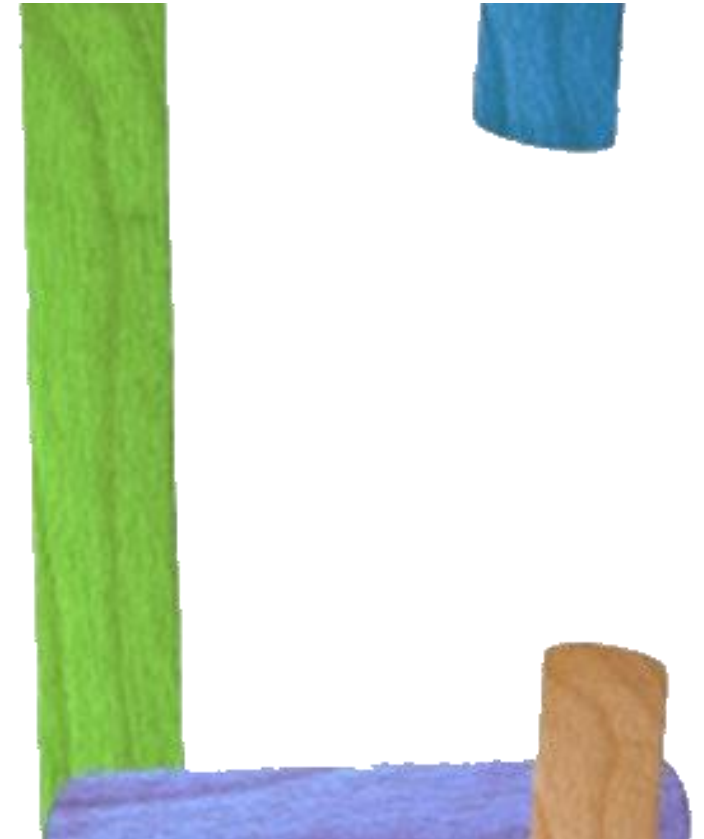
Poder: A linguagem C oferece controle de baixo nível sobre o hardware do computador, permitindo que os programadores escrevam código otimizado e de alto desempenho. Ela suporta ponteiros, aritmética de ponteiros e manipulação direta de bytes e bits.



A linguagem C é uma linguagem de programação de propósito geral que foi originalmente desenvolvida por Dennis Ritchie nos anos 1970 no Bell Labs.

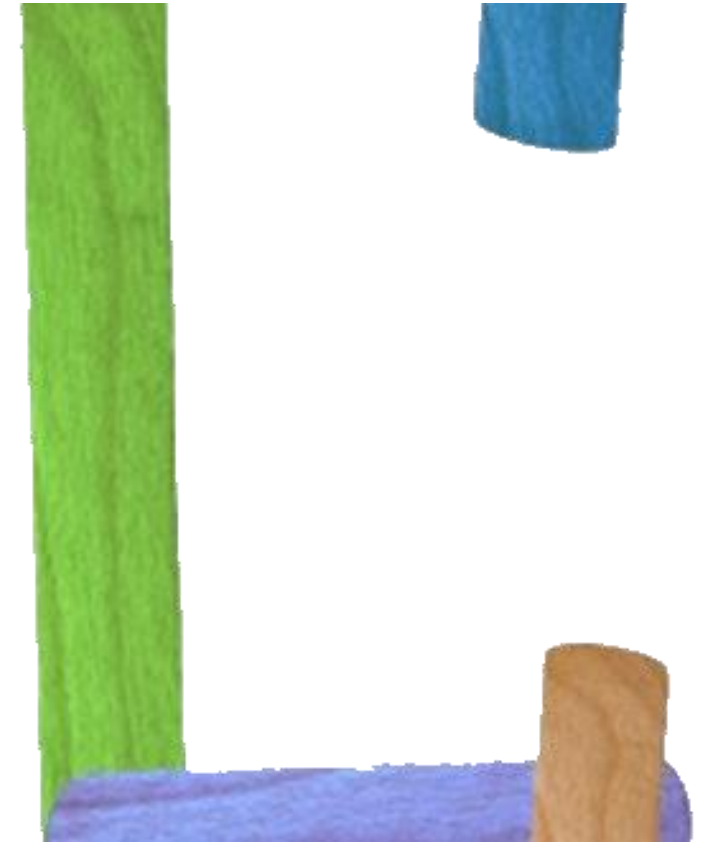
Modularidade: C suporta a criação de funções e a organização de código em módulos reutilizáveis. Isso permite que os programadores dividam um programa em partes menores, tornando-o mais fácil de entender, manter e depurar.

Bibliotecas Padrão: C possui uma biblioteca padrão robusta que fornece funções pré-definidas para realizar várias tarefas, como entrada e saída, manipulação de strings, alocação de memória, matemática, entre outros.



A linguagem C é uma linguagem de programação de propósito geral que foi originalmente desenvolvida por Dennis Ritchie nos anos 1970 no Bell Labs.

- +** : Soma dois operandos.
- : Subtrai o segundo operando do primeiro.
- *** : Multiplica dois operandos.
- /** : Divide o primeiro operando pelo segundo.
- %** : Retorna o resto da divisão inteira entre o primeiro e o segundo operando.



Linguagem C

Operadores Aritméticos

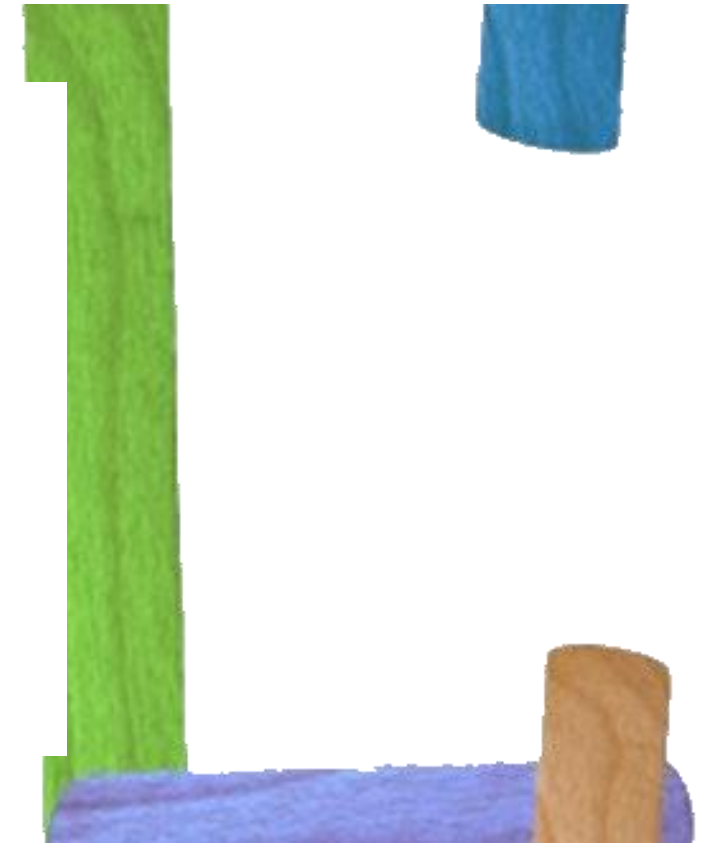
= : Atribui o valor do lado direito ao lado esquerdo.

+= : Incrementa o valor da variável da esquerda com o valor da direita e atribui o resultado à variável da esquerda.

-= : Decrementa o valor da variável da esquerda com o valor da direita e atribui o resultado à variável da esquerda.

***=** : Multiplica o valor da variável da esquerda pelo valor da direita e atribui o resultado à variável da esquerda.

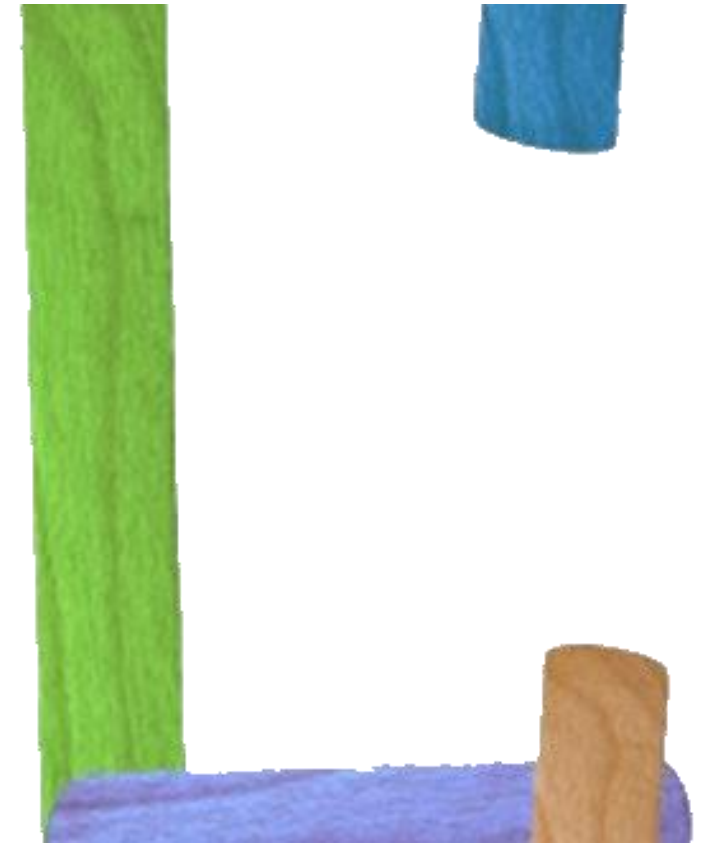
/= : Divide o valor da variável da esquerda pelo valor da direita e atribui o resultado à variável da esquerda.



Linguagem C

Operadores de Atribuição

- ==** : Verifica se dois operandos são iguais.
- !=** : Verifica se dois operandos são diferentes.
- <** : Verifica se o operando à esquerda é menor que o operando à direita.
- >** : Verifica se o operando à esquerda é maior que o operando à direita.
- <=** : Verifica se o operando à esquerda é menor ou igual ao operando à direita.
- >=** : Verifica se o operando à esquerda é maior ou igual ao operando à direita.



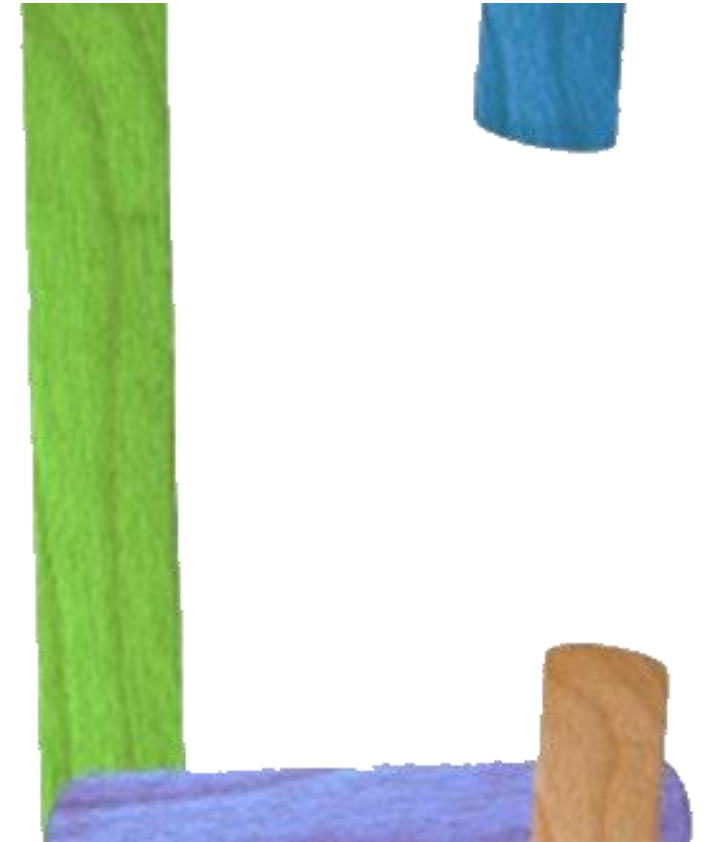
Linguagem C

Operadores de Comparação

&& : Retorna verdadeiro se ambos os operandos são verdadeiros.

|| : Retorna verdadeiro se pelo menos um dos operandos é verdadeiro.

! : Inverte o valor lógico do operando.



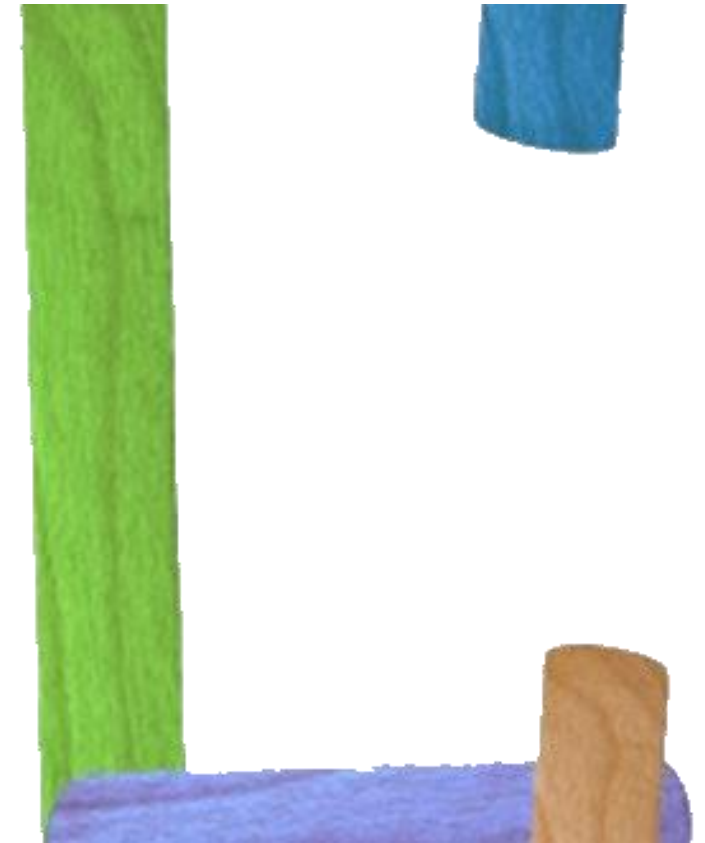
Linguagem C

Operadores Lógicos

++ : Incrementa o valor da variável por 1.

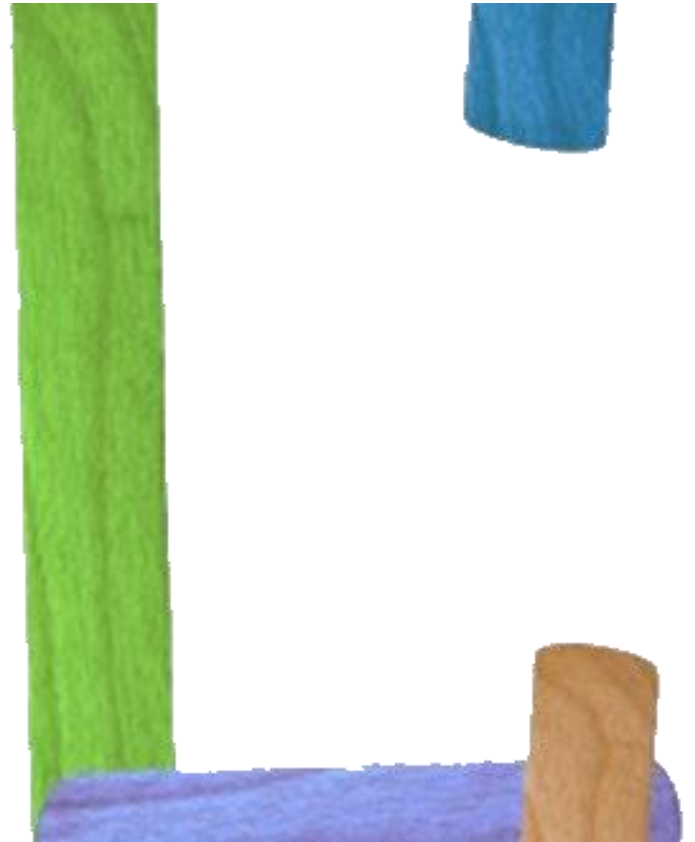
-- : Decrementa o valor da variável por 1.

Linguagem C



Operadores de Incremento
E Decremento

- &** : Realiza uma operação "E" bit-a-bit em dois operandos.
- |** : Realiza uma operação "OU" bit-a-bit em dois operandos.
- ^** : Realiza uma operação "OU Exclusivo" bit-a-bit em dois operandos.
- <<** : Desloca bits para a esquerda.
- >>** : Desloca bits para a direita.



Linguagem C

Operadores de Bit a Bit

Linguagem C

```
int main() {  
    int idade = 25;  
    float altura = 1.75;  
    char inicial = 'J';  
    char nome[] = "João";  
  
    printf("Nome: %s\n", nome);  
    printf("Idade: %d\n", idade);  
    printf("Altura: %.2f\n", altura);  
    printf("Inicial: %c\n", inicial);  
  
    return 0;  
}
```

printf: Imprime uma saída formatada na saída padrão.

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    int idade;  
    printf("Digite a sua idade: ");  
    scanf("%d", &idade);  
    printf("A sua idade é: %d\n", idade);  
    return 0;  
}
```



scanf: Lê a entrada formatada da entrada padrão..



A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

%d ou **%i**: Exibe um número inteiro.

%f: Exibe um número de ponto flutuante.

%c: Exibe um caractere.

%s: Exibe uma string.

%u: Exibe um número inteiro sem sinal.

%ld ou **%lld**: Exibe um número inteiro longo ou longo longo.

%lu ou **%llu**: Exibe um número inteiro longo ou longo longo sem sinal.

%e ou **%E**: Exibe um número de ponto flutuante em notação científica.



Especificadores

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

`%x` ou `%X`: Exibe um número hexadecimal.

`%o`: Exibe um número octal.

`%p`: Exibe um ponteiro.

`%%`: Exibe o caractere '%' literalmente.

`%ld`, `%lld`, `%hd`: Modificadores para especificar o tamanho do inteiro (long, long long, short).

`%f`, `%e`, `%g`: Modificadores para controlar a formatação de números de ponto flutuante.

`%s`: Modificador para limitar a quantidade de caracteres a serem exibidos de uma string.

`%10d`, `%.2f`, `%7s`: Modificadores para especificar a largura do campo e a precisão.


`%zu` exibe um valor do tipo `size_t`.

Especificadores


A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    int ch;  
  
    printf("Digite um caractere: ");  
    ch = getchar();  
  
    printf("O caractere digitado foi: %c\n", ch);  
  
    return 0;  
}
```



getchar: Lê um caractere da entrada padrão...



A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    char ch = 'A';  
  
    printf("Exibindo o caractere usando putchar: ");  
    putchar(ch);  
  
    return 0;  
}
```

putchar: Escreve um caracter na saída padrão.

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    char ch;  
  
    printf("Pressione uma tecla: ");  
    ch = getch();  
  
    printf("A tecla pressionada foi: %c\n", ch);  
  
    return 0;  
}
```

getch: Lê um caracter da entrada padrão sem exibir na tela

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    char ch = 'A'; // Caractere a ser exibido  
  
    putchar(ch); // Exibe o caractere usando a função putchar  
  
    return 0;  
}
```

putchar: Escreve um caracter na saída padrão sem quebra de linha

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    char buffer[100];  
  
    printf("Digite uma linha de texto: ");  
    gets(buffer);  
  
    printf("A linha digitada foi: %s\n", buffer);  
  
    return 0;  
}
```

gets: Lê uma linha de texto da entrada padrão e armazena em uma string

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    char buffer[100];  
  
    printf("Digite uma linha de texto: ");  
    fgets(buffer, sizeof(buffer), stdin);  
  
    printf("A linha digitada foi: %s", buffer);  
  
    return 0;  
}
```

fgets: Lê uma linha de texto da entrada padrão e armazena em uma string.

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
int main() {  
    char mensagem[] = "Hoje está a chover!!!";  
    puts(mensagem);  
  
    return 0;  
}
```

puts: Escreve uma string na saída padrão, seguida de uma quebra de linha.

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
void limparBufferEntrada() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {}
    fflush(stdin);
}

int main() {
    int numero;

    printf("Digite um número inteiro: ");
    scanf("%d", &numero);

    limparBufferEntrada();

    printf("Digite uma string: ");
    char texto[100];
    fgets(texto, sizeof(texto), stdin);

    printf("Número: %d\n", numero);
    printf("Texto: %s\n", texto);

    return 0;
}
```

getchar() e fflush(): Para limpar o buffer de entrada em C

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

```
void limparBufferEntrada() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {}
    fflush(stdin);
}

int main() {
    int numero;

    printf("Digite um número inteiro: ");
    scanf("%d", &numero);

    limparBufferEntrada();

    printf("Digite uma string: ");
    char texto[100];
    fgets(texto, sizeof(texto), stdin);

    printf("Número: %d\n", numero);
    printf("Texto: %s\n", texto);

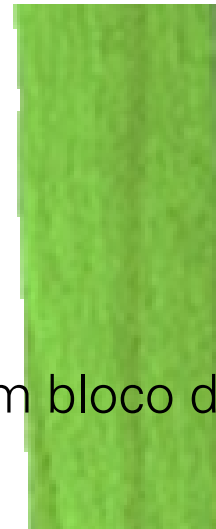
    return 0;
}
```

getchar() e fflush(): Para limpar o buffer de entrada em C

A linguagem C possui instruções built-in (internas) que fornecem funcionalidades específicas.

Linguagem C

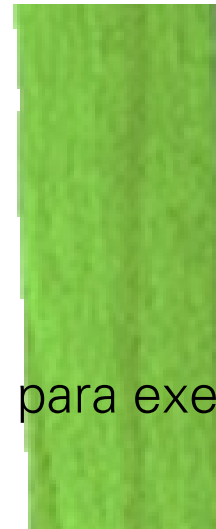
```
int main() {  
    int idade;  
  
    printf("Digite sua idade: ");  
    scanf("%d", &idade);  
  
    if (idade >= 18) {  
        printf("Você é maior de idade.\n");  
    } else {  
        printf("Você é menor de idade.\n");  
    }  
  
    return 0;  
}
```



A estrutura condicional **if** em C é usada para executar um bloco de código se uma determinada condição for verdadeira

Linguagem C

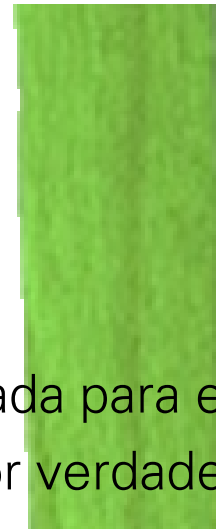
```
int main() {  
    int contador;  
  
    for (contador = 1; contador <= 5; contador++) {  
        printf("%d ", contador);  
    }  
  
    printf("\n");  
  
    return 0;  
}
```



A estrutura de controle de repetição **for** em C é utilizada para executar um bloco de código um número específico de vezes.

Linguagem C

```
int main() {  
    int contador = 1;  
  
    while (contador <= 5) {  
        printf("%d ", contador);  
        contador++;  
    }  
  
    printf("\n");  
  
    return 0;  
}
```



A estrutura de controle de repetição **while** em C é utilizada para executar repetidamente um bloco de código enquanto uma condição especificada for verdadeira

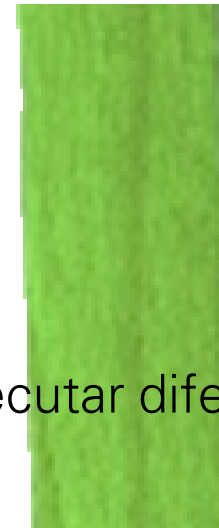
Linguagem C

```
int main() {  
    int numero;  
  
    do {  
        printf("Digite um número positivo: ");  
        scanf("%d", &numero);  
    } while (numero <= 0);  
  
    printf("Número digitado: %d\n", numero);  
  
    return 0;  
}
```



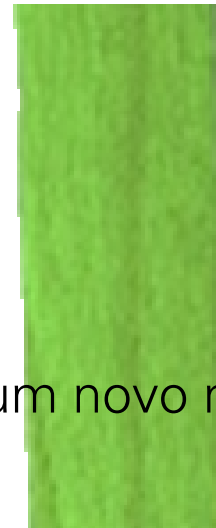
A estrutura de controle de repetição **do-while** em C é utilizada para executar um bloco de código pelo menos uma vez e, em seguida, repetidamente executá-lo enquanto uma determinada condição for verdadeira


```
int main() {  
    int dia;  
    printf("Digite o número do dia da semana (1 a 7): ");  
    scanf("%d", &dia);  
    switch (dia) {  
        case 1:  
            printf("Domingo\n");  
            break;  
        case 2:  
            printf("Segunda-feira\n");  
            break;  
        case 3:  
            printf("Terça-feira\n");  
            break;  
        case 4:  
            printf("Quarta-feira\n");  
            break;  
        case 5:  
            printf("Quinta-feira\n");  
            break;  
        case 6:  
            printf("Sexta-feira\n");  
            break;  
        case 7:  
            printf("Sábado\n");  
            break;  
        default:  
            printf("Número de dia inválido\n");  
            break;  
    }  
  
    return 0;  
}
```



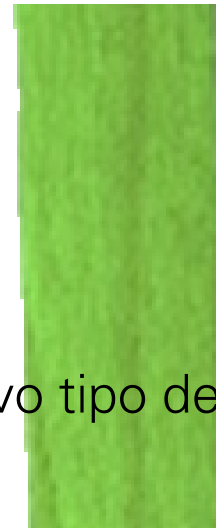
A estrutura de controle **switch** em C é utilizada para executar diferentes blocos de código com base no valor de uma expressão específica


```
typedef struct {  
    char nome[50];  
    int idade;  
} Pessoa;  
  
int main() {  
    Pessoa pessoal;  
  
    printf("Digite o nome da pessoa: ");  
    scanf("%s", pessoal.nome);  
  
    printf("Digite a idade da pessoa: ");  
    scanf("%d", &pessoal.idade);  
  
    printf("Nome: %s\n", pessoal.nome);  
    printf("Idade: %d\n", pessoal.idade);  
  
    return 0;  
}
```



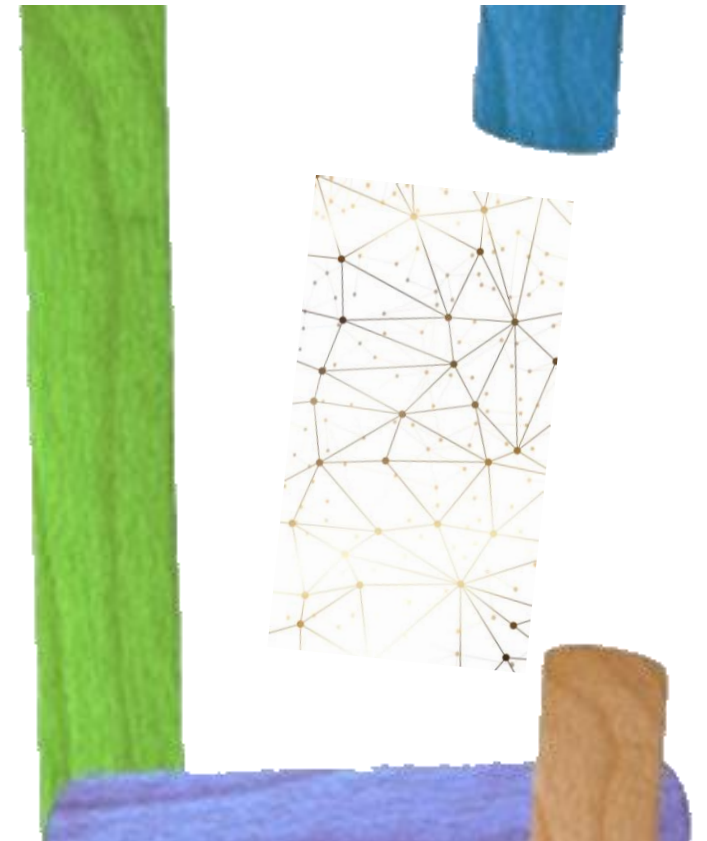
A palavra-chave **typedef struct** em C é usada para criar um novo nome (um tipo definido pelo utilizador) para um tipo de dados existente

```
enum DiaSemana {  
    Domingo,  
    Segunda,  
    Terca,  
    Quarta,  
    Quinta,  
    Sexta,  
    Sabado  
};  
  
int main() {  
    enum DiaSemana hoje = Terca;  
  
    if (hoje == Domingo || hoje == Sabado) {  
        printf("É fim de semana!\n");  
    } else {  
        printf("Não é fim de semana.\n");  
    }  
  
    return 0;  
}
```



A palavra-chave **enum** em C é utilizada para criar um novo tipo de dados enumerado, que consiste em um conjunto de constantes nomeadas

32 palavras reservadas			
auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while



As **palavras reservadas em C** são termos que têm significado especial na linguagem e são usados para declarar estruturas de controle, tipos de dados, funções e outras construções da linguagem. Essas palavras têm um significado pré-definido e **não podem ser usadas** como identificadores (nomes de variáveis, funções, etc.) pelo programador.