

Introdução a Linguagem C ANSI

PeD Talks...



Introdução

Na década de 1940, a computação estava em um estágio inicial, e os programadores escreviam os códigos diretamente em *linguagem* de máquina, que consistia em instruções numéricas compreendidas pelos computadores da época. Entretanto, essa abordagem era complexa e propensa a erros, exigindo um entendimento profundo da arquitetura do hardware.

Alguns dos pioneiros e principais contribuidores:



John Backus: Fortran

Em 1957, liderou a equipe que desenvolveu

Grace Hopper: CoBOL

Em torno de 1959, criadora da linguagem

Bjarne Stroustrup: C++

Nos anos 1980, criou a extensão da linguagem C

John McCarthy: LiSP

Por volta de 1958, ele criou a linguagem

Dennis Ritchie: Linguagem C

Na década de 1970, ele criou a linguagem

Sun Microsystems:

A equipe que desenvolveu a linguagem Java, lançada em 1995



Arquivos e Bibliotecas

Em linguagem C temos algumas `bibliotecas` padrão, o conjunto de `bibliotecas` é descrito em um arquivo-interface (grupo de dados de controle logicamente relacionados), nessas `bibliotecas` contém um conjunto básico de operações matemáticas, manipulação de cadeia de caracteres, conversão de tipos de dados e entrada e saída de `arquivo` e da tela.

As principais são:

- `stdlib` - Funções de conversão numérica, funções de geração de números aleatórios, alocação de memória, funções de controle de processo.
- `stdio.h` - Define as principais funções de entrada e saída.
- `strings.h` - Funções de manipulação de string.
- `locale.h` - Funções de localização.

Declaração de variáveis



Uma `variável` é uma posição de memória que pode ser identificada através de um nome. Podem ter seu conteúdo alterado por um comando de atribuição.

Podemos identificá las em três locais:

- Fora de todas as funções do programa, que torna as variáveis globais e podem ser usadas a partir de qualquer lugar no programa.
- Início de um bloco de código. Estas variáveis são chamadas **locais** e só têm validade dentro do bloco no qual são declaradas.
- Lista de parâmetros de uma função.

tipo	limite inferior	limite superior	código de formatação					
			número	caractere	caracteres	octal	hexa	notação científica
bool	0 false	1 true	%d			%o	%x	
char	-128	127	%d	%c	%s	%o	%x	
unsigned char	0	255	%d			%o	%x	
short int	-32.768	32.767	%d			%o	%x	
unsigned short int	0	65.535	%d			%o	%x	
int	-2.147.483.648	2.147.483.647	%d			%o	%x	
unsigned int	0	4.294.967.295	%lld			%o	%x	
long long int	-9.223.372.036.854.775.808	9.223.372.036.854.775.807	%lld			%o	%x	
unsigned long long int	0	18.446.744.073.709.551.616	%lld			%o	%x	
float	1.1×10^{-38}	3.4×10^{38}	%f					%e
double	2.2×10^{-308}	1.7×10^{308}	%lf					%le
long double	3.4×10^{-4932}	1.1×10^{-4933}	%Lf					%Le

Operadores aritméticos

Operador	Descrição	A = 10 B = 20	Resultado
+	Soma	A + B	30
-	Subtração	A - B	-10
*	Multiplicação	A * B	200
/	Divisão inteira	A / B	2
%	Módulo	B % A	0
++	Incremento	A++	11
--	Decremento	A-	9

Operadores relacionais



Operador	Descrição	A = 10 B = 20	Resultado
<code>==</code>	Igual	<code>A == B</code>	FALSE
<code>!=</code>	Diferente	<code>A != B</code>	TRUE
<code>></code>	Maior que	<code>A > B</code>	FALSE
<code><</code>	Menor que	<code>A</code>	2
<code>>=</code>	Maior igual	<code>A >= B</code>	FALSE
<code><=</code>	Menor Igual	<code>A <= B</code>	TRUE

Operadores lógicos

Operadores de atribuição

Operador	Descrição	A = 10 B = 20	Resultado
&&	E Lógico	A && B	FALSE
	Ou Lógico	A B	TRUE
!	Não Lógico	!A	FALSE

Operadores	Descrição
=	atribuição simples de valor a variável
+=	atribuição com soma
-=	atribuição com subtração
*=	atribuição com produto
/=	atribuição com divisão
%=	atribuição com módulo

Vetores e strings

O **vetor** é uma estrutura de dados indexada, que pode armazenar uma determinada quantidade de valores do mesmo tipo.

Os dados armazenados em um **vetor** são chamados de item do **vetor**.

Para localizar a posição de um índice em um vetor usamos um número item denominado índice do **vetor**.

Vantagem de utilização do **vetor**:

Facilidade de manipular um grande conjunto de dados do mesmo tipo declarando-se apenas uma variável.

O vetor é uma estrutura que armazena itens, cada item tem uma posição específica que são denominado índice, o índice é iniciado na posição 0.

G	u	s	t	a	v	o
0	1	2	3	4	5	6

O termo **strings** serve para identificar uma sequência de caracteres, na prática são usados para representar textos, na linguagem C, não existe um tipo de dados string nativo, para representar uma **string** em C temos que criar um **vetor** do tipo **char**.

Exemplo básico de declaração de string.

1 | `#include<stdio.h>` →

Inclusão da biblioteca.

2 |

3 |

4 | `int main(void){` →

Função principal "main".

5 |

6 | `char nome[100];` →

Declarando variáveis.

7 | `int i;`

8 |

9 | `printf("Digite seu nome: ");`

10 | `scanf("%s", &nome);`

11 | `printf("O nome armazenado foi: %s\n", nome);`

12 |

13 | `for (i=0; nome[i] != '\0'; i++){`

14 | `printf("valor do elemento %d da string = %c\n", i, nome[i]);` →

15 | `}`

16 |

17 | `return 0;` →

18 | `}`

Aqui finalizamos o nosso programa com "return 0" e fechando chaves da função, ou seja, se o programa for bem executado, logo não mostrará erros.

Sequências de printf e scanf para pedir e receber o nome do usuário, getch esta função armazena tudo que foi digitado, inclusive os espaços, até que a tecla ENTER seja pressionada.

Adicionando estrutura de repetição FOR e mostrando cada índice do vetor

Digite seu nome: gustavo

0 nome armazenado foi: gustavo

valor do elemento 0 da string = g

valor do elemento 1 da string = u

valor do elemento 2 da string = s

valor do elemento 3 da string = t

valor do elemento 4 da string = a

valor do elemento 5 da string = v

valor do elemento 6 da string = o

Process exited after 11.83 seconds with return value 0

Pressione qualquer tecla para continuar. . .

Iniciando com matrizes

Matrizes são basicamente vetores em duas ou mais dimensões, formando uma tabela na memória.

Os itens devem ser todos do mesmo tipo de dado.

Declarando matrizes:

```
float Media[5][2];
```

As matrizes devem ser lida primeiro a linha depois a coluna e as posições da tabela começam no número 0.

Para fazer o preenchimento de uma matriz, devemos percorrer todos os seus elementos e atribuir-lhes um valor.

Isto pode ser feito tanto gerando valores para cada elemento da matriz, como recebendo os valores pelo teclado.

Um método interessante para percorrer uma matriz é usar duas estruturas de repetição for e duas variáveis inteiras, uma para a linha e a outra para a coluna.

0	1	2	3	4
0	1	2	3	4

Sobre ponteiros

Ponteiros ou apontadores, são variáveis que armazenam o endereço de memória de outras variáveis.

Os ponteiros podem apontar para qualquer tipo de variável. Portanto temos ponteiros para int, float, double, etc.

Ponteiros são fundamentais quando precisando acessar a mesma variável em diferentes partes do programa.

Exemplos de uso:

- Alocação dinâmica de memória
- Manipulação de arrays.
- Para retornar mais de um valor em uma função.
- Referência para listas, pilhas, árvores e grafos.

Sintaxe básica sobre ponteiros

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

Biblioteca padrão

```
int main(void) {
```

Função principal

```
int a = 10  
int *ponteiro;  
ponteiro = &a;  
int b = 20;  
*ponteiro = b;
```

Declarando o ponteiro, é necessário o asterisco, para indicar que a variável é um ponteiro.

```
printf("%d %d\n", a, b);
```

Printf mostrando a variável a e b com seus respectivos valores.

```
return(0);
```

```
}
```

Estrutura de controle de fluxo

As estruturas de controle de fluxo desempenham um papel crucial na modelagem do comportamento de um programa, fornecendo as ferramentas necessárias para direcionar sua execução com base em condições específicas.



Controle Condicional

IF

O **if** permite que você execute um bloco de código se uma condição for verdadeira

ELSE

O **else** é opcional e permite que você execute um bloco de código alternativo se a condição do **if** não for atendida.

SWITCH

A estrutura **switch** é uma alternativa ao **if-else** e é usada para tomar decisões com base no valor de uma expressão



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

int main()
{
    int x;

    printf("Informe um numero: ");
    scanf("%d", &x);

    if(x > 10){
        printf("o numero eh maior que 10");
    }
    else{
        printf("O numero e menor que 10");
    }
    return 0;
}
```



Controle de repetição (Laços)

WHILE

O **while** executa um bloco de código repetidamente enquanto uma condição for verdadeira.

DO-WHILE

O **do-while** é semelhante, mas garante que o bloco de código seja executado pelo menos uma vez, pois a condição é verificada após a primeira execução.

FOR

A **estrutura for** é usada para criar loops quando você sabe antecipadamente quantas vezes deseja executar um bloco de código.



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

int main() {
    int i;

    printf("De 1 ate 10: ");

    while(i <= 10) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

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

int main() {
    int z, num;

    printf("Informe o numero para que conte ate ele\n");
    printf("Informe o numero: ");
    scanf("%d", &num);
    z = 1;

    do{
        printf("\n%d\n", z);
        z++;
    }while(z <= num);
    return 0;
}
```

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

```
int main(){
    int i;
```

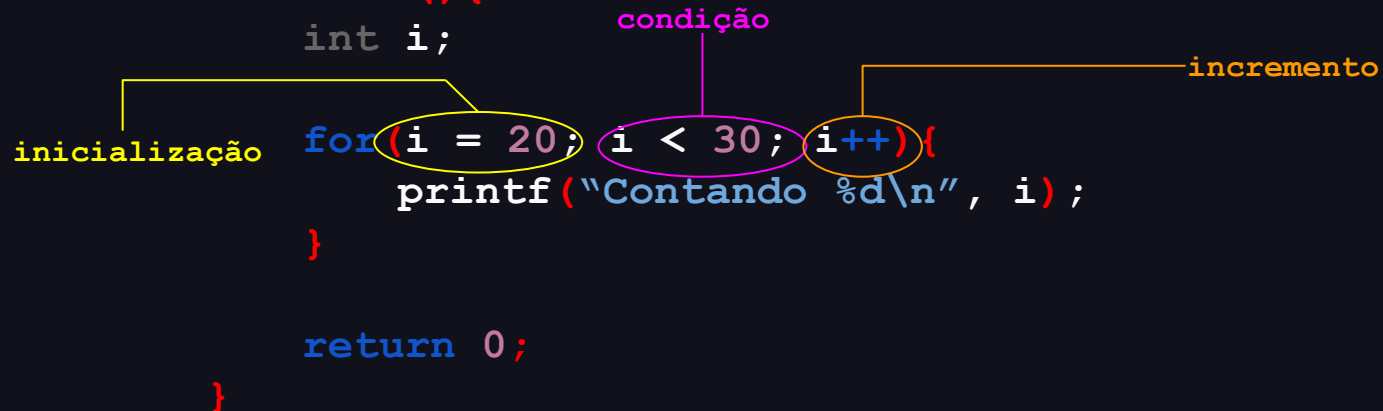
condição

inicialização

incremento

```
    for(i = 20; i < 30; i++){
        printf("Contando %d\n", i);
    }

    return 0;
}
```





Instruções de Controle de Loop

BREAK

O **break** é usado para interromper imediatamente a execução do loop e sair dele.

CONTINUE

O **continue** é usado para pular a iteração atual do loop e continuar com a próxima iteração, ignorando o código restante dentro do loop na iteração atual.



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

int main() {
    char opcao;

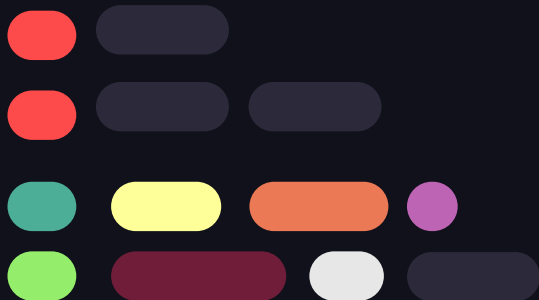
    while (1) {
        printf("Informe sua cidade: F - Fortaleza, I - Itaitinga: \n");
        opcao = toupper(getche());

        switch(opcao) {
            case 'F':
                printf("Voce eh de Fortaleza\n");
                break;
            case 'I':
                printf("Voce eh de Itaitinga\n");
                break;
            default:
                printf("Opcao invalida. Tente novamente.\n");
                continue;
        }

        break; // Saia do loop se a opção for válida
    }

    return 0;
}
```

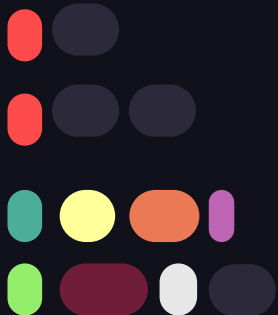

Funções





Benefícios de se usar Funções

- Permite o reaproveitamento de código já construído (por você ou por outros programadores);
- Evita que um mesmo trecho de código seja repetido várias vezes dentro de um mesmo programa e, com isso, qualquer alteração é feita apenas nesse trecho e de forma simples.
- Para que os blocos do programa não fiquem grandes demais e por consequência, mais difíceis de entender;
- Facilita a leitura do programa de maneira que os blocos de código possam ser logicamente compreendidos de forma isolada.



Declaração de Função

Para declarar uma função, você precisa fornecer:

1. Nome da função.
2. Tipo de retorno (int, float, void, etc.).
3. Parâmetros que a função aceita.

Sintaxe da função:

```
tipo_de_retorno nome_da_função (lista de parâmetros) {  
    instruções;  
    retorno_da_função;  
}
```

Definição da Função

A definição de uma função em linguagem c envolve a implementação do corpo da função, onde a lógica específica da tarefa que a função realiza é detalhada.

Sintaxe da definição da função:

```
tipo_de_retorno nome_da_funcao (tipo_do_parametro1 parametro1, tipo_do_parametro2
parametro2...){

    // Corpo da função - lógica específica aqui

    return valor_de_retorno; // Opcional, dependendo do tipo de retorno
}
```

Chamada de Função

A chamada de uma função em linguagem C ocorre quando você utiliza o nome da função seguido pelos argumentos necessários entre parênteses.

Sintaxe da Chamada de função:

```
tipo_de_retorno resultado = nome_da_funcao(arg1, arg2, ...);
```

Parâmetros de Função

A lista de parâmetros, também é chamada de lista de argumentos. Funcionam como a interface de comunicação (passagem de valores/dados) entre o programa (chamador) e a função. Os parâmetros de uma função são definidos como se estivesse declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas.

Ex:

Tipo do valor que será
retornado pela função

Declaração dos parâmetros da
função

Int nome_da_função(int num1, int num2)



Parâmetros de Função

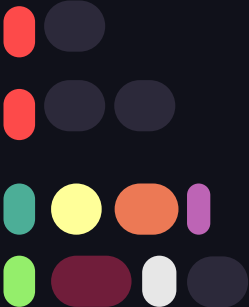
Existem 2 maneiras de passar dados através dos parâmetros: `Por valor` ou `Por referência`.

Por Valor:

Em C, os parâmetros são passados por valor por padrão. Isso significa que uma cópia dos valores é passada para a função, e qualquer modificação dentro da função não afeta as variáveis originais fora dela.

Por Referência:

Passar por referência significa que a função recebe um ponteiro que aponta para a localização de memória da variável original. Qualquer alteração feita através do ponteiro dentro da função afetará a variável original fora da função.



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

//Função soma
int soma(int x, int y){
    return x + y;
}

//Função principal
int main(void){
    int a;

    a = soma(10, 11);

    printf("Soma: %d",
a);

    return 0;
}
```

Definição da função

Parâmetros

Valor a retornar

Chamada da função

Thanks!

