Introdução à

Rodrigo Ferreira

Material para o curso:

https://github.com/RodrigoFerreira001/MinicursoC

Origem

C é uma linguagem e de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pela ISO, criada em 1972, por Dennis Ritchie e Brian Kernighan, no AT&T Bell Labs, para desenvolver o sistema operacional Unix.

C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para C. C tem influenciado muitas outras linguagens de programação, mais notavelmente C++, que originalmente começou como uma extensão para C.

Por que usar C?

- Sistemas operacionais
- Compiladores de linguagem
- Montadores
- Editores de texto
- Spoolers de impressão
- Drivers de rede
- Programas de Otimização
- Bases de dados
- Intérpretes de idioma
- Trabalhos Práticos(AEDS I,II e III)

Visão Geral e Compilação

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
   /* meu primeiro programa em C */
   printf("Hello, World! \n");

   return 0;
}
```

Para compilar:

gcc main.c -o main

Para executar:

./main

Sintaxe Básica

Ponto e vírgula:

```
printf("Hello, World! \n");
return 0;
```

Comentários:

```
// meu primeiro programa em C e isto é um comentário de linha

/* isto é um
comentário
 em bloco */
```

Chaves e Escopo:

```
int i = 0;
for(i; i < 10; i++){
    int a = 10;
    printf("%d\n", a);
}
```

Sintaxe Básica

Identificadores:

- Um identificador é usado para identificar uma variável, função ou outro item definido pelo usuário.
- Um identificador pode começar com uma letra A-Z, a-z ou “_”, seguido de zero ou mais letras, números e “_”.
- C é case-sensitive, ou seja, “count” é diferente de “Count” ou “COUNT”.

Exemplo:

- Correto: `int count;`
- Incorreto: `int 0123Count;`

Sintaxe Básica

Palavras Reservadas:

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

Variáveis

“Objeto (uma posição, frequentemente localizada na memória) capaz de reter e representar um valor ou expressão. Enquanto as variáveis só "existem" em tempo de execução, elas são associadas a "nomes", chamados identificadores, durante o tempo de desenvolvimento.”

Em C:

<tipo> <identificador>;

<tipo> <identificador1>, <identificador2>, …, <identificadorN>;

<tipo> <identificador> = <valor>;

Exemplo:

```
int i = 0;
char a = 'a';
```

Tipos Primitivos de Dados

Tipos Inteiros:

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Tipos Primitivos de Dados

Tipos de Pontos Flutuantes:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Tipo Booleano:

C não possui tipos booleanos, neste caso, qualquer valor diferente de 0 e NULL é considerado **verdadeiro**, caso contrário, **falso**.

Tipos Primitivos de Dados

Tipo Void: Tipo “vazio”, pode ocorrer em três situações:

- **Funções que retornam void**:
  Não necessita conter o “return” ao final do escopo.

- **Funções que recebem void**:
  Não aceita nenhum argumento.

- **Variáveis do tipo void***:
  Variáveis genéricas, podem ser “convertidas” para qualquer tipo.

Input/Ouput em C

Biblioteca <stdio.h>

Leitura:

```
scanf("máscara", &variável);
```

Escrita:

```
printf("máscara", valor1, ..., valorN);
```

Input/Ouput em C

Máscaras:

%c char single character
%d (%i) int signed integer
%e (%E) float or double exponential format
%f float or double signed decimal
%g (%G) float or double use %f or %e as required
%o int unsigned octal value
%p pointer address stored in pointer
%s array of char sequence of characters
%u int unsigned decimal
%x (%X) int unsigned hex value

Prática 0

Operadores Aritméticos

Assumindo duas variáveis, A e B, com A valendo 10 e B valendo 20:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11

Operadores Relacionais

Assumindo duas variáveis, A e B, com A valendo 10 e B valendo 20:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Operadores Lógicos

Assumindo duas variáveis, A e B, com A valendo 1 e B valendo 0:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Operadores “Bit-a-bit”

Assumindo duas variáveis, A e B, com A valendo 13 e B valendo 60:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Operadores de Atribuição

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2

Operadores Diversos

Operator	Description
sizeof	sizeof operator returns the size of a variable. For example, sizeof(a), where a is integer, will return 4.
Condition ? X : Y	Conditional operator. If Condition is true ? then it returns value X : otherwise value Y
,	Comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
. (dot) and -> (arrow)	Member operators are used to reference individual members of classes, structures, and unions.
Cast	Casting operators convert one data type to another. For example, int(2.2000) would return 2.
&	Pointer operator & returns the address of an variable. For example &a; will give actual address of the variable.
*	Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var.

Precedência de Operadores

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<<>>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Prática 1

Estruturas de Decisão: If-Else

Sintaxe:

```
if(condição){
    //Ação se verdadeiro
}else{
    //Ação se falso
}
```

Estruturas de Decisão: Switch-Case

Sintaxe:

```
switch(valor):
    case 0:
        //Ação
    break;

    case 1:
        //Ação
    break;

    ...

    case N:
        //Ação
    break;

    default:
        //Ação
    break;
}
```

Estruturas de Decisão: Operador Ternário

Sintaxe:

```
condição ? caso verdadeiro : caso falso;
```

Quando usar cada um?

Prática 2

Estruturas de Repetição

For:

```
for(valor inicial; condição; passo){
    //Instrução
}
```

While:

```
while(condição){
    //Instrução
}
```

Do-While:

```
do{
    //Instrução
}while(condição);
```

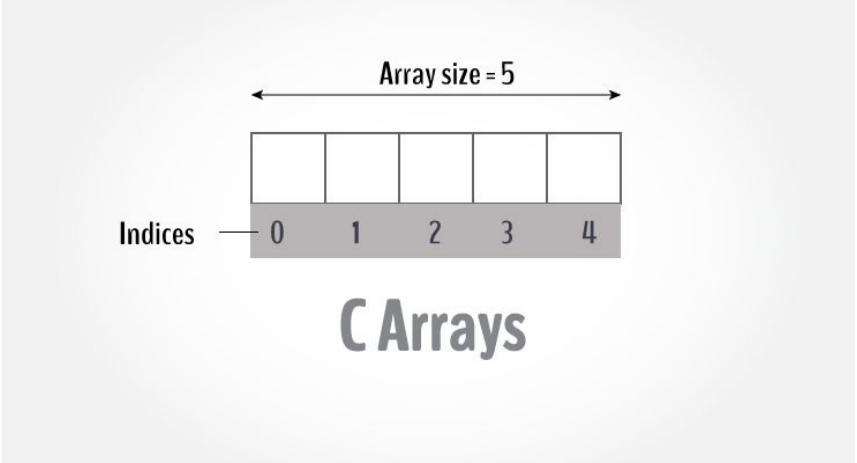
Quando usar cada um?

Prática 3

Vetores(Arrays) e Matrizes

“Estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, um índice ou uma chave.”

Obs: No C, índices começam no 0.



Vetores(Arrays) e Matrizes

Sintaxe:

<tipo> <identificador>[tamanho1]...[tamanhoN];

ou

<tipo> <identificador>[] = {valor1, valor2, …, valorN};

Exemplo:

```
int vetor[5];
int vetor2[] = {1,2,3,4,5,6,7};
```

Vetores(Arrays) e Matrizes

Acesso dos valores:

variável[indice] = valor;

ou

printf(“%d”, variável[indice]);

Funções e Procedimentos

“Consiste em uma porção de código que resolve um problema muito específico, parte de um problema maior.”

Função: Há retorno de valor.

Procedimento: Não há retorno de valor (void).

Sintaxe:

```
<tipo de retorno> <nome> (<lista de parâmetros>){
	//Instruções
}
```

Estruturas e Typedef

Estruturas(Structs): “Permitem que dados relacionados sejam combinados e manipulados como um todo, aumentando o nível de abstração.”

Sintaxe:

```
struct <nome>{
    <tipo> <identificador>;
    …
};
```

Estruturas e Typedef

Exemplo:

```c
//Declaração
struct ponto{
    int x;
    int y;
};

//uso
struct ponto p1;
p1.x = 10;
p1.y = 0;
```

Podemos melhorar a abstração?

Estruturas e Typedef

Typedef: Permite ao programador definir um novo nome para um determinado tipo.

Sintaxe:

typedef <tipo existente> <apelido>;

Exemplo:

```
typedef int batatinha;


batatinha a = 10;
```

Estruturas e Typedef

Typedef + Struct:

```
typdef struct ponto{
    int x;
    int y;
}Ponto;

Ponto a;
a.x = 10;
a.y = 0;
```

Prática 4

Prática Final da Parte Introdutória

Exercício: Construa um programa que calcule o IMC(Índice de Massa Corporal) de uma pessoa. O programa deve solicitar ao usuário sua altura(float), peso(float) e seu sexo. Baseado na tabela abaixo, o programa deve informar ao usuário em qual situação ele se encontra.

IMC - Índice de Massa Corporal	HOMEM	MULHER
Obesidade Mórbida	+ de 43	+ de 39
Obesidade Moderada	30 a 39,9	29 a 38,9
Obesidade Leve	25 a 29,9	24 a 28,9
Normal	20 a 24,9	19 a 23,9
Abaixo do Normal	- de 20	- de 19

Fórmula de Cálculo do IMC - Índice de Massa Corporal
Peso (em Kgr.) + Altura ^{2} (em metros) = IMC

Dúvidas?

Parte prática

- Argumentos
- Ponteiros, Referências e Operador de Seta (->)
- Alocação e Matriz Dinâmica
- Funções com Matrizes e Arrays
- Modularização
- Manipulação de Arquivos
- Bibliotecas string.h
- Atoi, atof

Argumentos


```c
int main(int argc, char *argv[]) {
   /* meu primeiro programa em C */
   printf("Hello, World! \n");

   return 0;
}
```

argc: “argument count”, informa o número de argumentos recebidos.

argv: “argument vector”, “vetor de strings”, os argumentos em si.

Argumentos

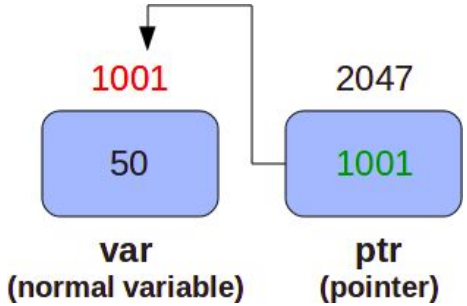
```c
int main(int argc, char *argv[]) {
   printf("Número de argumentos: %d\n", argc);
   printf("Argumentos:\n");

   int i = 0;
   for(i = 0; i < argc; i++){
       printf("%s\n",argv[i]);
   }

   return 0;
}
```

Prática 5

Ponteiros

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

```
int var = 50;
int *ptr = &var;
```



Sintaxe:

<tipo> *<identificador>;

Ponteiros

Acessar o valor de um ponteiro: *

```c
int var = 50;
int *ptr = &var;
printf("Valor de ptr: %d", *ptr);
```

Structs:

```c
Ponto *p = (Ponto *) malloc(sizeof(Ponto));
(*p).x = 10;
(*p).y = 0;

//Simplificando
printf("p.x = %d", p->x);
printf("p.y = %d", p->y);
```

Prática 6

Alocação Dinâmica

“Reserva de espaço de memória em tempo de execução.”

Vantagens:

- Controle e otimização do programa;
- Programas mais plásticos;
- Permite o uso de TADs

Alocação Dinâmica

Sintaxe:

<tipo> *<identificador> = (tipo *) malloc(sizeof(tipo));

Como funciona?

A função **malloc** reserva um espaço de memória do tamanho retornado pela função **sizeof,** e retorna esse endereço para a variável do tipo ponteiro;

Alocação Dinâmica: Arrays

Sintaxe:

<tipo> *<identificador> = (tipo *) malloc(sizeof(tipo) * <tamanho_vetor>);

Como acessar?

***(variavel + posição) = …**

ou

variavel[posição] = ...

Alocação Dinâmica: Matrizes

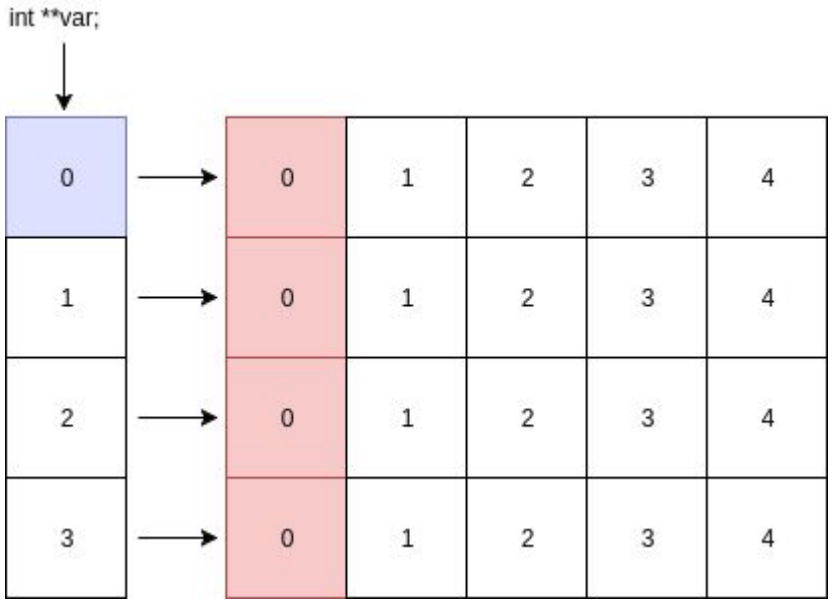
Sintaxe:

```
<tipo> **<identificador> = (tipo **) malloc(sizeof(tipo *) * <linhas>);
for(i = 0; i < linhas; i++){
    <identificador>[i] = (tipo *) malloc(sizeof(tipo) * <colunas>);
}
```

Acesso?

```
<identificador>[i][j] = ...
```

Alocação Dinâmica: Matrizes



Alocação Dinâmica: Limpeza

Variáveis simples:

```
free(var);
```

Arrays:

```
free(var);
```

Matrizes:

```
for(i = 0; i < linhas; i++){
    free(var[i]);
}

free(var);
```

Matrizes e Arrays em Funções

Como usar?

Array: void func(int vec[]){...}
Matriz: void func(int matriz[][colunas]){...}

ou

Array: void func(int *vec){...}
Matriz: void func(int **matriz){...}

Prática 7

Modularização

“Compõe o ferramental necessário para um programa mais legível com uma melhor manutenção e melhor desempenho por meio da programação estruturada.”

“Dividir as responsabilidades em pequenos módulos.”

Prática: Modularização

Arquivos

Permite a persistência de dados entre as execuções
Sintaxe:

FILE *arquivo = fopen(<nome>, <modo>);

onde o modo pode ser:

r - open for reading
w - open for writing (file need not exist)
a - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)

Prática: Arquivos

Strings

“Array de caracteres.”

Sintaxe:

```
char nome[50];
```

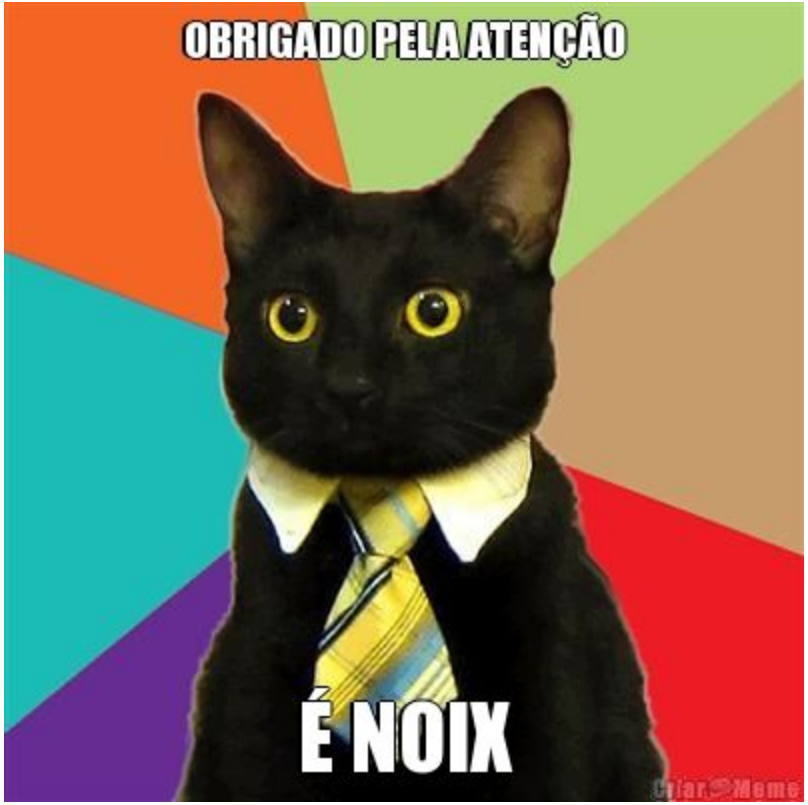
Prática: Strings

Dúvidas?

Material de Apoio

- **Tutorials Point:** https://www.tutorialspoint.com/cprogramming/
- **Cpluplus.com:** http://www.cplusplus.com/reference/

Livro:
C Completo e Total, HERBERT SCHILDT


OBRIGADO PELA ATENÇÃO
É NOIX
Criar Meme