Algoritmos e Lógica de Programação e Conceitos básicos

Alexandre Mello

Fatec Campinas

2023

Roteiro

Estrutura Básica de um Programa em C

A estrutura básica é a seguinte:

```
Declaração de bibliotecas Usadas

Declaração de variáveis

int main(){
Declaração de variáveis

Comandos
.
.
.
Comandos
}
```

Estrutura Básica de um Programa em C

Exemplo:

```
#include <stdio.h>

int main(){
  int a;
  int b,c;

  a = 7+9;
  b = a+10;
  c = b-a;
}
```

Variáveis

Definição

Variáveis são locais onde armazenamos valores. Toda variável é caractereizada por um nome, que a identifica em um programa, e por um tipo, que determina o que pode ser armazenado naquela variável.

- Durante a execução do programa, um pedacinho da memória corresponde à variável.

Declarando uma variável

Declara-se da seguinte forma: **Tipo_Variável Nome_Variável;**

Exemplos corretos:

- int soma;
- float preco_abacaxi;
- char resposta;

Exemplos incorretos:

- soma int;
- float preco_abacaxi

Variáveis inteiras

Variáveis utilizadas para armazenar valores inteiros. Ex: 13 ou 1102 ou 24.

Abaixo temos os **tipos da linguagem C** que servem para armazenar inteiros:

- **int:** Inteiro cujo comprimento depende do processador. É o inteiro mais utilizado. Em processadores Intel comum, ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647.
- **unsigned int:** Inteiro cujo comprimento depende do processador e que armazena somente valores positivos. Em processadores Intel comum, ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295.

Variáveis inteiras

- **long int:** Inteiro que ocupa 64 bits em computadores Intel de 64bits, e pode armazenar valores de aprox. -9×10^{18} a aprox. 9×10^{18}.
- **unsigned long int:** Inteiro que ocupa 64 bits em computadores Intel de 64bits, e armazena valores de 0 até aprox. 18×10^{18}.
- **short int:** Inteiro que ocupa 16 bits e pode armazenar valores de -32.768 a 32.767.
- **unsigned short int:** Inteiro que ocupa 16 bits e pode armazenar valores de 0 a 65.535.

Variáveis inteiras

Exemplos de declaração de variáveis inteiras:

- int numVoltas;
- int ano;
- unsigned int quantidadeChapeus;

Exemplos Inválidos:

- int int numVoltas;
- unsgned int ano;

Variáveis inteiras

Você pode declarar várias variáveis de um mesmo tipo. Basta separar as variáveis por vírgula:

- int numVoltas , ano;
- unsigned int a, b, c, d;

Variáveis de tipo caractere

Variáveis utilizadas para armazenar letras e outros símbolos existentes em textos. OBS: Guarda apenas um caractere.

Exemplos de declaração:

- char umaLetra;
- char YorN;

Variáveis de tipo ponto flutuante

Armazenam valores reais. Mas possuem problemas de precisão pois há uma quantidade limitada de memória para armazenar um número real. Exemplos de números em ponto flutuante: 2.1345 ou 9098.123.

- **float:** Utiliza 32 bits, e na prática tem precisão de aproximadamente 6 casas decimais (depois do ponto). Pode armazenar valores de $(+/-)10^{-38}$ a $(+/-)10^{38}$
- **double:** Utiliza 64 bits, e na prática tem precisão de aproximadamente 15 casas decimais. Pode armazenar valores de $(+/-)10^{-308}$ a $(+/-)10^{308}$

Variáveis de tipo ponto flutuante

Exemplos de declaração de variáveis de tipo ponto flutuante.

- float salario;
- float resultado, cotacaoDolar;
- double a, b, c;

Regras para nomes de variáveis em C

- **Deve** começar com uma letra (maíuscula ou minúscula) ou subcrito(_). **Nunca** pode começar com um número.
- Pode conter letras maíusculas, minúsculas, números e subscrito.
- Não pode-se utilizar como parte do nome de uma variável:

```
{ ( + - * / \ ; . , ?
```

- Letras maiúsculas e minúsculas são diferentes:

```
int c;
int C;
```

Regras para nomes de variáveis em C

As seguintes palavras já tem um significado na linguagem C e por esse motivo não podem ser utilizadas como nome de variáveis:

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

Comando de Atribuição

Definição

O comando de atribuição serve para atribuir valores para variáveis.

- A sintaxe do uso do comando é:

variável = valor ;

- Exemplos:

```
int a;
float c;
a = 5;
c = 67.89505456;
```

Comando de Atribuição

- O comando de atribuição pode conter expressões do lado direito:

variável = expressão ;

- Atribuir um valor de uma expressão para uma variável significa calcular o valor daquela expressão e copiar aquele valor para a variável.

Comando de Atribuição

No exemplo abaixo, a variável **soma** recebe o valor calculado da expressão **a + b**.

```
soma = a + b;
```

Variável

Expressão

Comando de Atribuição

- Exemplos:

```
int a;
float c;
a = 5 + 5 + 10;
c = 67.89505456 + 8 - 9;
```

Atribuição

- O sinal de igual no comando de atribuição é chamado de **operador de atribuição**.
- Veremos outros operadores mais adiante.

À esquerda do operador de atribuição deve existir somente o nome de uma **variável**.

$=$

À direita, deve haver uma **expressão** cujo valor será calculado e armazenado na variável.

Variáveis e Constantes

Constantes são valores previamente determinados e que por algum motivo, devem aparecer dentro de um programa.

- Assim como as variáveis, as constantes também possuem um tipo. Os tipos permitidos são exatamente os mesmos das variáveis, mais o tipo `string`, que corresponde a uma sequência de caracteres.
- Exemplos de constantes:

```
85, 0.10, 'c', "Hello, world!"
```

Variáveis e Constantes

- Uma **constante inteira** é um número inteiro como escrito normalmente
  Ex: `10`, `145`, `1000000`
- Uma **constante ponto flutuante** é um número real, onde a parte fracionária vem depois de um ponto
  Ex: `2.3456`, `32132131.5`, `5.0`
- Uma **constante do tipo caractere** é sempre representada por um caractere (letra, dígito, pontuação, etc.) entre aspas simples.
  Ex: `'A'`, `'!'`, `'4'`, `'('`
- Uma **constante do tipo string** é um texto entre aspas duplas
  Ex: `"Hello, world!"`

Expressões Simples

Uma constante é uma expressão e como tal, pode ser atribuída a uma variável (ou ser usada em qualquer outro lugar onde uma expressão seja válida).

- Ex1:

```
int a;
a = 10;
```

- Ex2:

```
char b;
b = 'F';
```

- Ex3:

```
double c;
c = 3.141592;
```

Expressões Simples

Uma variável também é uma expressão e pode ser atribuída a outra variável.

Ex:

```
int a, b;
a = 5;
b = a;
```

Exemplos de atribuição

- **OBS:** A declaração de uma variável sempre deve ocorrer antes de seu uso.

```
int a,b;
float f;
char h;

a = 10;
b = -15;
f = 10.0;
h = 'A';

a = b;
f = a;
a = (b+a);
```

Qual o valor final na variável **a**?

Exemplos errados de atribuição

```
int a,b;
float f,g;
char h;

a b = 10; //Errado! Por quê?
b = -15
d = 90;  //Errado! Por quê?
```

Exercício

Qual o valor armazenado na variável **a** no fim do programa?

```
int main(void){
    int a, b, c, d;

    d = 3;
    c = 2;
    b = 4;
    d = c + b;
    a = d + 1;
    a = a + 1;

}
```

Exercício

Compile o programa abaixo? Você sabe dizer qual erro existe neste programa?

```
int main(void){
    int a, b;
    double c,d;
    int g;

    d = 3.0;
    c = 2.4142;
    b = 4;
    d = b + 90;
    e = c * d;
    a = a + 1;

}
```

Informações Extras: Constantes Inteiras

- Um número inteiro como escrito normalmente
  Ex: `10`, `145`, `1000000`
- Um número na forma hexadecimal (base 16), precedido de 0x
  Ex: `0xA` ($0xA_{16} = 10$), `0x100` ($0x100_{16} = 256$)
- Um número na forma octal (base 8), precedido de 0
  Ex: `010` ($0x10_8 = 8$)

Informações Extras: Constantes do tipo de ponto flutuante

- Na linguagem C, um número só pode ser considerado um número decimal se tiver uma parte "não inteira", mesmo que essa parte não inteira tenha valor zero. Utilizamos o ponto para separarmos a parte inteira da parte "não inteira".
  Ex: `10.0`, `5.2`, `3569.22565845`
- Um número inteiro ou decimal seguido da letra **e** mais um expoente. Um número escrito dessa forma deve ser interpretado como:

$$numero \cdot 10^{expoente}$$

Ex: 2e2 ($2e2 = 2 \cdot 10^2 = 200.0$)

Informações Extras: caractere

- São, na verdade, variáveis inteiras que armazenam um número associado ao símbolo. A principal tabela de símbolos utilizada pelos computadores é a tabela ASCII (*American Standard Code for Information Interchang*), mas existem outras (EBCDIC, Unicode, etc ..).
- `char`: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de -128 a 127.
- `unsigned char`: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de 0 a 255.
- Toda constante do tipo caractere pode ser usada como uma constante do tipo inteiro. Nesse caso, o valor atribuído será o valor daquela letra na tabela ASCII.

Informações Extras: Tabela ASCII

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 16	caracterees de Controle															
32		!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	/	]	^	_
96	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	—	}	~	

Informações Extras: Obtendo o tamanho de um tipo

O comando `sizeof(tipo)` retorna o tamanho, em bytes, de um determinado tipo. (Um byte corresponde a 8 bits).

Exemplo

```
printf ("%d", sizeof(int));
```

Escreve 4 na tela.

Escrevendo na tela

- Para imprimir um texto, utilizamos o comando **printf**. O texto pode ser uma constante do tipo string.

Exemplo

```
printf("Olá Pessoal!");
```
Saída: Olá Pessoal!

- No meio da constante string pode-se incluir caracteres de formatação especiais. O símbolo especial `\n` é responsável por pular uma linha na saída.

Exemplo

```
printf("Olá Pessoal!  \n Olá Pessoal");
```
Saída: Olá Pessoal!
Olá Pessoal

Escrevendo o conteúdo de uma variável na tela

- Podemos imprimir, além de texto puro, o conteúdo de uma variável utilizando o comando **printf**. Para isso utilizamos símbolos especiais no texto, para representar que aquele trecho deve ser substituído por uma variável ou constante, e no final, passamos uma lista de variáveis ou constantes, separadas por vírgula.

Exemplo

```
int a=10;
printf("A variável %s contém o valor %d","a", a);
```

Saída: A variável a contém o valor 10

- Nesse caso, %s deve ser substituído por uma variável ou constante do tipo **string**, enquanto %d deve ser substituído por uma variável ou constante do tipo **int**.

Formatos inteiros

%d — Escreve um inteiro na tela.

Exemplo

```
printf ("%d", 10);
Saída:  10
```

Exemplo

```
int a=12;
printf ("O valor e %d", a);
Saída: O valor e 12
```

Formatos inteiros

- A letra d pode ser substituída pelas letras u e ld, quando desejamos escrever variáveis do tipo unsigned ou long, respectivamente.

Exemplo

printf ("%d", 4000000000);
Saída: -294967296.
Enquanto que
printf ("%ld", 4000000000);
Saída: 4000000000.

Formatos ponto flutuante

%f — Escreve um ponto flutuante na tela.

Exemplo

printf ("%f", 10.0);
Saída: 10.000000

Formatos ponto flutuante

%.*N*f — Escreve um ponto flutuante na tela, com *N* casas decimais.

Exemplo

```
printf ("%.2f", 10.1111);
```

Saída: 10.11

Formatos ponto flutuante

- O formato %f pode ser substituido por %lf, para escrever um **double** ao invés de um **float**.

Exemplo

printf ("%.2lf", 10.0);
Saída: 10.00

Formato caracter

%c — Escreve um caracter.

Exemplo

printf ("%c", 'A');
Saída: A

Note que **printf ("%c", 65)** também imprime a letra A. Por quê?

Formato **string**

%s — Escreve uma string

Exemplo

printf ("%s", "Meu primeiro programa");
Saída: Meu primeiro programa

A função **scanf**

- Realiza a leitura de dados a partir do teclado.
- Parâmetros:
  - Uma string, indicando os tipos das variáveis que serão lidas e o formato dessa leitura.
  - Uma lista de variáveis.
- Aguarda que o usuário digite um valor e atribui o valor digitado à variável.

A função **scanf**

O programa abaixo é composto de quatro passos:

1. Cria uma variável **n**
2. Escreve na tela "Digite um número:".
3. Lê o valor do número digitado.
4. Imprime o valor do número digitado.

```
#include <stdio.h>
int main(){
  int n;
  printf("Digite um número: ");
  scanf("%d",&n);
  printf("O valor digitado foi %d\n",n);
}
```

Formatos de leitura de variável

Os formatos de leitura são muito semelhantes aos formatos de escrita utilizados pelo **printf**. A tabela a seguir mostra alguns formatos possíveis de leitura

Código	Função
%c	Lê um único caracter
%s	Lê uma série de caracteres
%d	Lê um número decimal
%u	Lê um decimal sem sinal
%ld	Lê um inteiro longo
%f	Lê um número em ponto flutuante
%lf	Lê um double

A função **scanf**

O programa abaixo, lê um caracter, depois um número ponto flutuante e por fim um decimal. Por fim o programa imprime os dados lidos.

```
#include <stdio.h>

int main(){
  char c;
  float b;
  int a;

  printf("Entre com um caracter:");
  scanf("%c", &c);
  printf("Entre com um ponto flutuante:");
  scanf("%f", &b);
  printf("Entre com um número:");
  scanf("%d",&a);

  printf("Os dados lidos foram: %c, %f, %d \n",c,b,a);
}
```

Note que no **scanf**, cada variável para onde será lido um valor, deve ser precedida do caracter **&**.

Expressões

- Já vimos que constantes e variáveis são expressões.
- Uma expressão também pode ser um conjunto de operações aritméticas, lógicas ou relacionais utilizadas para fazer "cálculos" sobre os valores das variáveis.

Exemplo

a + b
Calcula a soma de **a** e **b**.

Expressões Aritméticas

- Os operadores aritméticos são: +, -, *, /, %
- *expressão* + *expressão* : Calcula a soma de duas expressões.
  Ex: 10 + 15;
- *expressão* - *expressão* : Calcula a subtração de duas expressões.
  Ex: 5 - 7;
- *expressão* * *expressão* : Calcula o produto de duas expressões.
  Ex: 3 * 4;

Expressões

- *expressão* / *expressão* : Calcula a divisão de duas expressões.
  Ex: 4 / 2;
- *expressão* % *expressão* : Calcula o resto da divisão (inteira) de duas expressões.
  Ex: 5 % 2;
- - *expressão* : Inverte o sinal da expressão.
  Ex: -5;

Expressões

Mais sobre o operador resto da divisão: %

- Quando computamos "a dividido por b", isto tem como resultado um valor p e um resto $r < b$ que são únicos tais que

$$a = p * b + r$$

- Ou seja a pode ser dividido em p partes inteiras de tamanho b, e sobrará um resto $r < b$.

Exemplos:
5%2 tem como resultado o valor 1.
15%3 tem como resultado o valor 0.
1%5 tem como resultado o valor 1.
19%4 tem como resultado o valor 3.

Expressões

No exemplo abaixo, quais valores serão impressos?

```
#include <stdio.h>

int main(){

  printf("%d \n", 27%3);
  printf("%d \n", 4%15);
}
```

Expressões

Mais sobre o operador /

- Quando utilizado sobre valores inteiros, o resultado da operação de divisão será inteiro. Isto significa que a parte fracionária da divisão será desconsiderada.
  - 5/2 tem como resultado o valor 2.
- Quando pelo menos um dos operandos for ponto flutuante, então a divisão será fracionária. Ou seja, o resultado será a divisão exata dos valores.
  - 5.0/2 tem como resultado o valor 2.5.

Expressões

No exemplo abaixo, quais valores serão impressos?

```
#include <stdio.h>

int main(){
  int a=5, b=2;
  float c=5.0, d=2.0;

  printf("%d \n",a/b);
  printf("%f \n", a/d);
  printf("%f \n", c/d);
}
```

Expressões

- As expressões aritméticas (e todas as expressões) operam sobre outras expressões.
- É possível compor expressões complexas como por exemplo:
  `a = b *( (2 / c)+(9 + d * 8) );`

Qual o valor da expressão 5 + 10 % 3?
E da expressão 5 * 10 % 3?

Precedência

- Precedência é a ordem na qual os operadores serão avaliados quando o programa for executado. Em C, os operadores são avaliados na seguinte ordem:
  - * e /, na ordem em que aparecerem na expressão.
  - %
  - + e -, na ordem em que aparecerem na expressão.
- Exemplo: 8+10*6 é igual a 68.

Alterando a precedência

- (*expressão*) também é uma expressão, que calcula o resultado da expressão dentro dos parênteses, para só então calcular o resultado das outras expressões.
  - $5 + 10 \ \% \ 3$ é igual a 6
  - $(5 + 10) \ \% \ 3$ é igual a 0
- Você pode usar quantos parênteses desejar dentro de uma expressão.
- Use sempre parênteses em expressões para deixar claro em qual ordem a expressão é avaliada!

Incremento(++) e Decremento(−−)

- É muito comum escrevermos expressões para incrementar/decrementar o valor de uma variável por 1.

```
a = a + 1;
```

- Em C, o operador unário ++ é usado para incrementar de 1 o valor de uma variável.

```
a = a + 1; é o mesmo que a++;
```

- O operador unário −− é usado para decrementar de 1 o valor de uma variável.

```
a = a - 1; é o mesmo que a--;
```

Exercício

- Crie um programa que:
  - Lê um caracter, pula uma linha e imprime o caracter lido.
  - Lê um inteiro, pula uma linha e imprime o inteiro lido.
  - Lê um número ponto flutuante, pula uma linha e imprime o número lido.

Exercício

- Crie um programa que lê dois números **double** e que computa e imprime a soma, a diferença, a multiplicação e divisão dos dois números.

Informações Extras: Incremento(++) e Decremento(−−)

Há uma diferença quando estes operadores são usados à esquerda ou à direita de uma variável e fizerem parte de uma expressão maior:

- **++a** : Neste caso o valor de **a** será incrementado antes e só depois o valor de **a** é usado na expressão.
- **a++**: Neste caso o valor de **a** é usado na expressão maior, e só depois é incrementado.
- A mesma coisa acontece com o operador −−.

O programa abaixo imprime "b: 6".

```
#include <stdio.h>

int main(){
  int a=5, b, c;

  b = ++a;

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

Já o programa abaixo imprime "b: 5".

```
 #include <stdio.h>

int main(){
  int a=5, b, c;

  b = a++;

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

Informações Extras: Atribuições simplificadas

Uma expressão da forma

```
a = a + b
```

onde ocorre uma atribuição a uma das variáveis da expressão pode ser simplificada como

```
a += b
```

Informações Extras: Atribuições simplificadas

Comando	Exemplo	Corresponde a:
+=	a += b	a = a + b;
-=	a -= b	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Informações Extras: Conversão de tipos

- É possível converter alguns tipos entre si.
- Existem duas formas de fazê-lo: implícita e explícita:
- Implícita
  - Capacidade (tamanho) do destino deve ser maior que a origem senão há perda de informação.
    Ex.: `int a; short b; a = b;`
    Ex: `float a; int b=10; a = b;`
- Explícita:
  - Explicitamente informa o tipo que o valor da variável ou expressão é convertida.
    Ex. `a = (int)( (float)b / (float)c );`
  - Não modifica o tipo "real" da variável, só o valor de uma expressão.
    Ex. `int a; (float)a=1.0;` ← Errado

Informações Extras: Um uso da conversão de tipos

A operação de divisão (/) possui dois modos de operação de acordo com os seus argumentos: inteira ou de ponto flutuante.

- Se os dois argumentos forem inteiros, acontece a divisão inteira. A expressão `10 / 3` tem como valor 3.
- Se um dos dois argumentos for de ponto flutuante, acontece a divisão de ponto flutuante. A expressão `1.5 / 3` tem como valor `0.5`.

Quando se deseja obter o valor de ponto flutuante de uma divisão (não-exata) de dois inteiros, basta converter um deles para ponto flutuante:

Exemplo

A expressão `10 / (float) 3` tem como valor `3.33333333`

Informações Extras: comentários

- O código fonte pode conter comentários direcionados unicamente ao programador. Estes comentários devem estar delimitados pelos símbolos /* e */, e são ignorados pelo compilador.

Exemplo

```
#include <stdio.h>

/* Este é o meu primeiro programa. */
//Isto tambem é um comentário
int main() {
  printf("Hello, world!\n");
}
```

- Comentários são úteis para descrever o algoritmo usado e para explicitar suposições não óbvias sobre a implementação.