

Construindo um programa em C

Professores(as):

Virgínia Fernandes Mota

João Eduardo Montandon de Araujo Filho

Leandro Maia Silva

INTRODUÇÃO A PROGRAMAÇÃO - SETOR DE INFORMÁTICA



Agora que vocês tem uma noção de algoritmos básicos, vamos introduzir a linguagem de programação C:

- ❶ Estrutura geral de um Programa C;
- ❷ Variáveis, funções de entrada e funções de saída;
- ❸ Atribuições e operações matemáticas;
- ❹ Os elementos que contribuem para o estilo de um programa em C.

Todo programa em C começa pela execução da função **main**.

- ❶ O programa começa pela chamada da função **main**;
- ❷ A partir daí, os comandos e funções inseridos em **main** são executados em sequência.

Veremos como criar nossas próprias funções mais adiante no curso!

A função main

```
1 // Constante que associa o significado de sucesso ao valor 0
2 #define SUCESSO 0
3
4 /**
5  * O início do programa acontece aqui!.
6  * @param argc Número de argumentos.
7  * @param argv Valores dos argumentos.
8  * @return SUCESSO caso o programa termine corretamente, ou caso
9  * contrário, o código do erro ocorrido.
10 */
11 int main(int argc, char ** argv) {
12     ... comandos ...
13
14     // Se chegou até aqui é porque correu tudo bem
15     return SUCESSO;
16 }
```

A função main - Hello World

```
1 #include <stdio.h> // Para usar o printf
2
3 #define SUCESSO 0
4
5 int main(int argc, char ** argv){
6     printf("Estou aprendendo a programar em C, que emocao!!!\n");
7     return SUCESSO;
8 }
```

Como Executar

\$ Sintaxe: gcc arq1 arq2 ... arqn -o arquivo_de_saida

\$ gcc arquivo.c -o arquivo.exe

\$./arquivo.exe

- Geralmente uma operação bem mais complicada do que aquela que pode ser realizada por uma única instrução;
- Uma função pode ser pré-estabelecida pelo próprio compilador ou criada pelo programador;
- Após ser criada, uma função pode ser encarada como um comando criado pelo programador;
- Uma função pode referenciar outras funções já criadas pelo programador.

- O propósito de uma função deve ser bem definido;
- Toda função tem um nome. Esse nome é utilizado para acionar ou chamar a função:
 - `printf("printf eh uma funcao");`
- O nome da função é sempre seguido de parênteses. Os parâmetros da função aparecem entre esses parênteses:
 - `printf("printf eh uma funcao");`
- Uma função pode conter nenhum, um, ou vários parâmetros. Parâmetros são sempre separados por uma vírgula.

- Quando compilamos um programa, o compilador invoca um pré-processador, que associa um rótulo a um trecho de código ou a um valor;
- A diretiva **#include** <arquivo> instrui o compilador a inserir o trecho de código armazenado em arquivo
 - O arquivo onde está o trecho de código a ser incluído é chamado de arquivo cabeçalho;
 - Mais usados: **stdio.h**, **stdlib.h**, **math.h**, **string.h**;
 - Cada arquivo contém trechos de código que implementam funções bem específicas:
 - **math.h**: funções matemáticas;
 - **stdio.h**: funções que imprimem e que lêem dados;
 - **stdlib.h**: funções que implementam tarefas do sistema e manipulação de memória;
 - **string.h**: funções que manipulam vetores de caracteres como se fossem strings.

- Armazena um dado de um determinado tipo, que pode ser recuperado através de seu nome;
- Endereços de memória destinados a armazenar informações durante a execução do programa;
- **Nomenclatura:** Mesma de algoritmos!!
 - Deve iniciar com letra ou _;
 - não pode ter caracteres especiais;
 - não pode ser palavras reservadas da linguagem;
 - Contém essencialmente caracteres alfanuméricos.

Variáveis - Tipos Disponíveis

- Essencialmente, possui os mesmos tipos dos presentes em algoritmos.

Algoritmo	Linguagem em C
inteiro	int
real	float ou double
caractere	char
logico	-
string	char[] terminado em '\0'

- Por padrão, números sem sufixos e sem ponto são do menor tipo inteiro que ele caiba. Ex.: int: 1, 15, 33, 1985 / long int: 2200000000 / long long int 2200000000000.
- Por padrão, números sem sufixo e com ponto são do tipo double. Ex.: 0.3, 1.18. 19.85.

- Valores que não mudam no decorrer da execução do programa podem ser representados por constantes;
- A declaração é dada pela diretiva **#define** rótulo valor;
 - Exemplo: `#define PI 3.14`;
 - O pré-processador substitui todas as referências a PI pelo valor 3.14;
 - Geralmente o rótulo é definido em letras maiúsculas.

É claro que podemos utilizar variáveis ao invés de constantes, porém a variável ocupa mais memória. Não é necessário fazer isto para qualquer constante! Faça apenas para constantes que tenham um significado bem definido associado, tipo o SUCESSO.

Operador de atribuição

- O operador de atribuição em C é o sinal de igual =;
- Sintaxe: <variavel> = <expressão>;

```
1 int a, b, c = 0, d;  
2 a = 5;  
3 b = a;  
4 d = a + b - c;  
5 a = (a/2)*3;  
6 a++; // equivalente à a = a + 1  
7 a--; // equivalente à a = a - 1  
8 a += 3; // equivalente à a = a + 3  
9 a -= 3; // equivalente à a = a - 3  
10 a /= 2; // equivalente à a = a / 2  
11 a *= 2; // equivalente à a = a * 2
```

- Conversões automáticas de valores na avaliação de uma expressão.

```
1 int a, c;  
2 float b, d;  
3 a = 4.5; // conversão implícita de double para int  
4 b = a / 2.0; // conversão implícita de int pra double  
5 // e de double para float  
6 c = 1/2 + b; // conversão implícita de float para int  
7 d = 1.0 / 2.0 + b;  
8 // Valores: a = 4, b = 2.0, c = 2, d = 2.5
```

Conversão de Tipos - Cast

- é possível explicitamente fazer a conversão de tipos usando o operador **cast**;
- Sintaxe: **(tipo)** variável

```
1 int num1, num2, a;  
2 float resdiv;  
3 a = (int) 2.5;  
4 num1 = 5;  
5 num2 = num1;  
6 resdiv = (float) num1 / num2;
```

A saída de dados na tela pode ser realizada pela função `printf`. Existem códigos de conversão que nos permitem imprimir dados de diferentes tipos:

- `%c` -> para imprimir dados do tipo caractere (`char`);
- `%d` ou `%i` -> para imprimir dados do tipo inteiro (`int`);
- `%e` -> para imprimir em notação científica;
- `%s` -> para imprimir dados do tipo cadeia de caracteres (`string`)

Exemplos

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Estou aprendendo a programar em C");
5     return SUCESSO;
6 }
```

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Valor recebido foi %d", 10);
5     return SUCESSO;
6 }
```


Exemplos

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Caracter A: %c ", 'A');
5     return SUCESSO;
6 }
```

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     int x = 10;
5     printf("Valor inteiro o %d e um float %f ", x, 1.10);
6     printf("Outro valor float eh %f", 1.10 + 2.35);
7     return SUCESSO;
8 }
```

Impressão de caracteres especiais

Código	Ação
<code>\n</code>	leva o cursor para a próxima linha
<code>\t</code>	executa uma tabulação
<code>\b</code>	executa um retrocesso
<code>\f</code>	leva o cursor para a próxima página
<code>\a</code>	emite um sinal sonoro (beep)
<code>\“</code>	exibe o caractere “
<code>\\</code>	exibe o caractere \
<code>%%</code>	exibe o caractere %

Fixando casas decimais

- Por default, a maioria dos compiladores C exibem os números de ponto flutuante (reais - float) com seis casas decimais:
- Para alterar este número podemos acrescentar .n ao código de formatação da saída, sendo n o número de casas decimais pretendido.

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Default: %f \n", 3.1415169265);
5     printf("Uma casa: %.1f \n", 3.1415169265);
6     printf("Duas casas: %.2f \n", 3.1415169265);
7     printf("Três casas: %.3f \n", 3.1415169265);
8     printf("Notacao cientifica: %e \n", 3.1415169265);
9     return SUCESSO;
10 }
```

- O programa pode fixar a coluna da tela a partir da qual o conteúdo de uma variável, ou o valor de uma constante será exibido. Isto é obtido acrescentando-se um inteiro *m* ao código de formatação. Neste caso, *m* indicará o número de colunas que serão utilizadas para exibição do conteúdo.

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Valor: %d \n", 25);
5     printf("Valor: %10d \n", 25);
6     return SUCESSO;
7 }
```

- 1 Fazer um programa que imprima o seu nome;
- 2 Modificar o programa anterior para imprimir na primeira linha o seu nome, na segunda linha a sua idade e na terceira sua altura;
- 3 Imprimir o valor 2.346728 com 1, 2, 3 e 5 casas decimais.

Exercícios Resolvidos

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Meu nome");
5     return SUCESSO;
6 }
```

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf("Meu nome \n Minha idade \n Minha altura \n");
5     return SUCESSO;
6 }
```

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     printf(" %.1f \n", 2.346728);
5     printf(" %.2f \n", 2.346728);
6     printf(" %.3f \n", 2.346728);
7     printf(" %.5f \n", 2.346728);
8     return SUCESSO;
9 }
```

- **scanf();**
- Ela é o complemento de **printf()** e nos permite ler dados formatados da entrada padrão (teclado);
- Sintaxe: `scanf("expressão de controle", argumentos);`
- Os argumentos do `scanf` são endereços das variáveis onde os valores lidos serão armazenados.
- Para obter o endereço de uma variável, deve-se usar o operador `&` (e comercial).

Exemplos

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     int num;
5     printf("Digite um valor");
6     scanf("%d", &num);
7     printf("\n O valor digitado foi %d", num);
8     return SUCESSO;
9 }
```


Exemplos

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     int n1, n2, soma;
5     printf("Digite dois valores: ");
6     scanf("%d", &n1);
7     scanf("%d", &n2);
8     soma = n1 + n2;
9     printf("\n %d + %d = %d", n1, n2, soma);
10    return SUCESSO;
11 }
```

Exemplos

Exemplo1 Ler a temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit. A fórmula de conversão é $F = (9 * C + 160)/5$

```
1  Início
2      real: C, F;
3      imprime("Digite uma temperatura em celsius");
4      leia(C);
5      F = (9*C + 160) / 5;
6      imprime(C + "em celsius equivale a " + F + "em Fahrenheit");
7  Fim
```

```
1  #include <stdio.h>
2  #define SUCESSO 0
3  int main(int argc, char ** argv){
4      float C, F;
5      printf("Digite uma temperatura em celsius");
6      scanf("%f", &C);
7      F = (9*C + 160) / 5;
8      printf(" %f em celsius equivale a %f em fahrenheit.", C, F);
9      return SUCESSO;
10 }
```

Exemplos

Exemplo2 Calcular e apresentar o volume de uma lata de óleo cilíndrica, a partir da leitura do raio da base e da altura.

```
1 Início
2   real: R, H, V;
3   imprime( "Digite o raio da base e a altura");
4   leia(R);
5   leia(H);
6    $V = 3.14 * R * R * H$ ;
7   imprime( "O volume é ", V);
8 Fim
```

```
1 #include <stdio.h>
2 #define SUCESSO 0
3 int main(int argc, char ** argv){
4     float R, H, V;
5     printf("Digite o raio da base e a altura");
6     scanf("%f %f", &R, &H);
7      $V = 3.14 * R * R * H$ ;
8     printf("O volume eh %f: ", V);
9     return SUCESSO;
10 }
```

Exemplos

Exemplo3 Ler os valores do comprimento, da largura e da altura de uma caixa, calcular e imprimir o seu volume.

```
1  Início
2  real: C, H, V;
3  imprime( "Digite o comprimento, a largura e a altura de sua caixa");
4  leia(C);
5  leia(L);
6  leia(H);
7  V = C * L * H;
8  imprime( "O volume é ", V);
9  Fim
```

```
1  #include <stdio.h>
2  #define SUCESSO 0
3  int main(int argc, char ** argv){
4      float C, L, H, V;
5      printf("Digite o comprimento, a largura e a altura de sua caixa");
6      scanf("%f %f %f", &C, &L, &H);
7      V = C * L * H;
8      printf("O volume eh %f: ", V);
9      return 0;
10 }
```

Vamos adotar um estilo?!

Devemos adotar um estilo de escrita quando estamos programando?

Devemos adotar um estilo de escrita quando estamos programando?

Sim! O estilo vai depender de uma série de fatores:

- Se você for um estudante: o estilo vai facilitar que seu professor entenda seu código;
- Se você já trabalha: você terá que adotar o estilo imposto pela empresa.

A razão para adotarmos um estilo de programação está na facilidade de leitura.

- Serão vários programas e vários programadores;
- Se todos usarem o mesmo estilo, todos irão ler e entender melhor os programas.

O estilo universal

```
1 int main(){ int a; a = 20; printf("Imprimindo o valor de a: %d", a);  
2 return 0;}
```

```
1 #define SUCESSO 0  
2 int main(int argc, char ** argv){  
3     int a;  
4     a = 20;  
5     printf("Imprimindo o valor de a: %d", a);  
6     return SUCESSO;  
7 }
```

Todo bloco de comandos deve ser indentado

- Um programa bem feito não é apenas um conjunto de instruções;
- A documentação envolve todo texto que venha a descrever o que o programa faz;
- A documentação não deve ser feita após o programa estar pronto, mas sim à medida em que ele vai sendo desenvolvido;
- A documentação também deve conter o modo de uso do programa, bem como as simplificações assumidas pelo programador.

- Texto embutido no código-fonte, que ajuda no entendimento do programa;
- Regra geral: Sempre comentar trechos do programa que não são óbvias. Nunca comentar trechos do programa que sejam óbvios;
- Duas formas de inserir comentários:
 - ❶ `//linha comentada`
 - ❷ `/*trecho comentado*/`

- 1 Construir um algoritmo para ler 5 valores inteiros, calcular e imprimir a soma desses valores;
- 2 Construir um algoritmo para ler 6 valores reais, calcular e imprimir a média aritmética desses valores;
- 3 Fazer um algoritmo para gerar e imprimir o resultado do número H, sendo $H = 1 + 1/2 + 1/3 + 1/4 + 1/5$;
- 4 Calcular o aumento que será dado a um funcionário, obtendo do usuário as seguintes informações : salário atual e a porcentagem de aumento. Apresentar o novo valor do salário e o valor do aumento;
- 5 A nota final de um aluno é dada pela média ponderada das notas das provas. Sabendo que o professor deu 3 provas, com pesos 4, 3 e 3, respectivamente, calcule a nota final do aluno;