



Linguagem C

Referência - I

■ Estrutura de um programa em C

```
/*  
 * main.c  
 *  
 * Criado em: 15/02/2014  
 * Autor: marco  
 */  
#include <stdio.h>  
#include <stdlib.h>  
int main(void)  
{  
    printf("Primeiro programa\n");  
    return EXIT_SUCCESS;  
}
```

■ Estrutura de um programa em C

```
/*  
 * main.c  
 *  
 * Criado em: 15/02/2014  
 * Autor: marco  
 */
```

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

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

```
int main(void)
```

```
{
```

```
    printf ("
```

```
    return EXIT_SUCCESS
```

```
}
```

Primeiro alerta!

O compilador C é **sensível ao caso!** Não é como em Pascal!

Assim, os **nomes** main, Main, MAIN etc **são diferentes entre si!**

Todas as palavras-chave de C são escritas em **letras minúsculas.**

PRESTE ATENÇÃO AO DIGITAR!

■ Estrutura de um programa em C

```
/*  
 * main.c  
 *  
 * Criado em: 15/02/2014  
 * Autor: marco  
 */
```

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

```
int main(void)
```

```
{
```

```
    printf("Primeiro programa\n");
```

```
    return EXIT_SUCCESS;
```

```
}
```

Comentários são
anotações escritas em uma
ou mais linhas **delimitadas**
pelos **símbolos** /* e */

Estrutura do programa

■ Estrutura de um programa em C

```
/*  
 * main.c  
 *  
 * Criado em: 15/02/2019  
 * Autor: marco  
 */  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void)  
{  
    printf("Primeiro programa\n");  
    return EXIT_SUCCESS;  
}
```

`#include` é um **símbolo de pré-processamento** (processado antes da compilação) e serve para **incluir no programa declarações presentes em outros arquivos**.

Neste caso, deseja-se incluir declarações de **funções padrão** existentes nas bibliotecas:

`.stdio.h` → funções de **entrada e saída**;

`.stdlib.h` → funções **utilitárias**.


Arquivos com extensão “.h” são denominados **arquivos de cabeçalho** (*header files*).

Duas formas de inclusão:

- `#include <nome>` → biblioteca padrão
- `#include "nome"` → biblioteca não padrão

■ Estrutura de um programa em C

```
/*  
 * main.c  
 *  
 * Criado em: 15/02/2014  
 * Autor: marco  
 */  
#include <stdio.h>  
#include <stdlib.h>  
int main(void)  
{  
    printf("Primeiro programa\n");  
    return EXIT_SUCCESS;  
}
```



A **função** `main()` é o **ponto de partida** de qualquer **programa C**. **Todo programa** escrito em C possui **uma e única** função `main()`.

A **forma básica** não possui **parâmetro** (`void`) e **retorna** um **valor inteiro** (`int`) que:

- **se for zero**, sinaliza ao sistema operacional que o programa **terminou normalmente**;
- **se for diferente de zero**, sinaliza ao sistema operacional um **término anormal**.

Estrutura do programa

■ Estrutura de um programa em C

```
/*
 * main.c
 *
 * Criado em: 15/02/2014
 * Autor: marco
 */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("Primeiro programa\n");
    return EXIT_SUCCESS;
}
```

O símbolo “\n” é entendido como um **caractere de escape** – forma de **representar caracteres invisíveis** que possuem alguma função – neste caso, **sinaliza nova linha**.

O **corpo** de uma **função C** é **delimitado** por “{” e “}”.

Cada **comando** do corpo é **terminado** pelo **símbolo “;”**.

`printf()` é uma **função presente** em `stdio.h` e que **serve para exibir** na **saída padrão** (tela) uma **cadeia de caracteres** formatada.

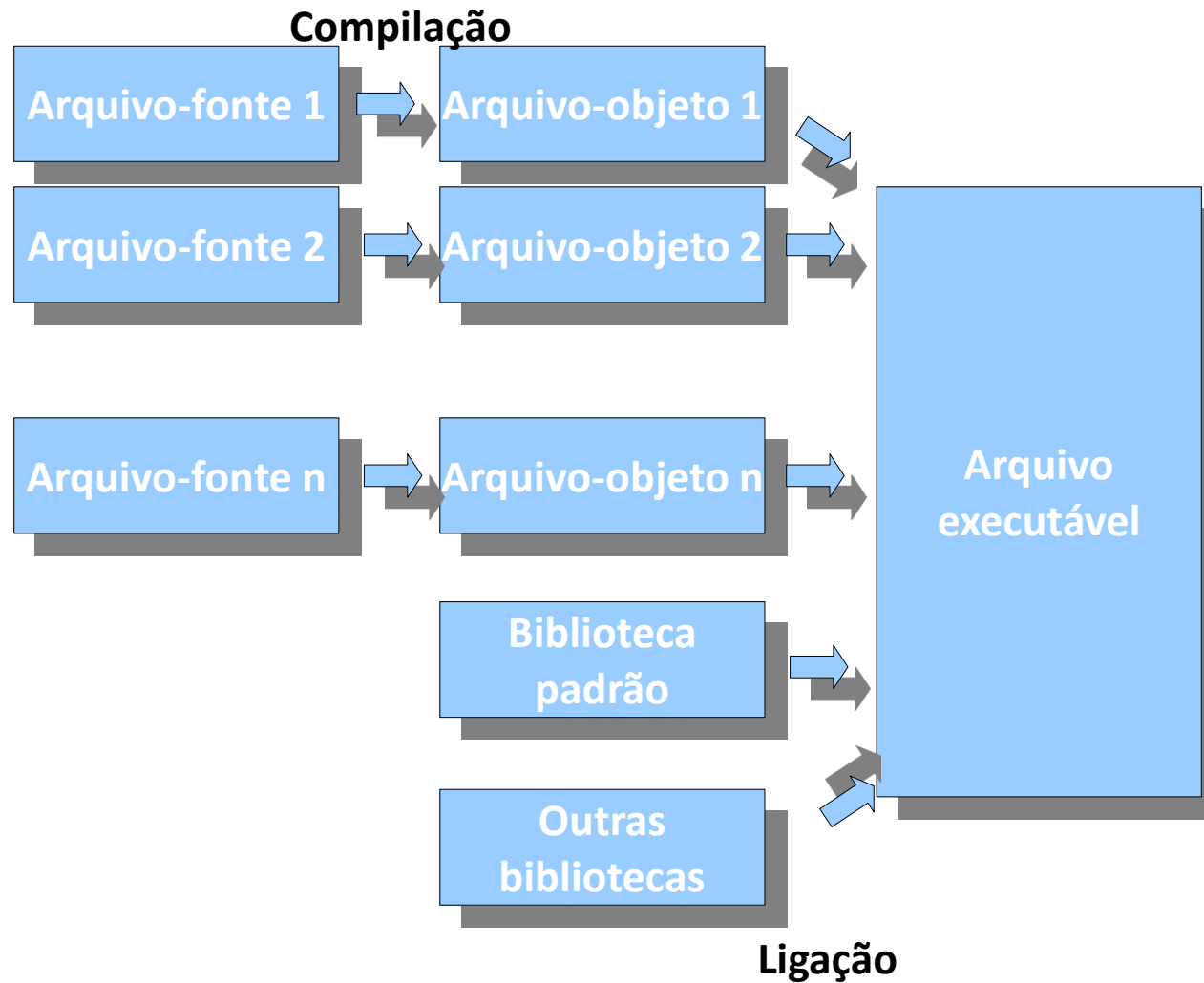
Literais de **cadeias** de caracteres são escritos dentro de **aspas duplas**.

`return` **termina** uma **função** **retornando** um **resultado** do **tipo declarado** para a **função** → neste caso, um **número inteiro**.

`EXIT_SUCCESS` é uma **constante (macro)** definida em `stdlib.h` e é um **valor igual à zero**.

Processo de compilação

■ Etapas



Declarações e tipos primitivos de dados

■ Declarações e atribuições de variáveis

```
float a, b, c;  
int d = 3, e;  
double r;  
a = 10;  
r = pow(a, d);
```

Em C **toda variável precisa ser declarada** antes de seu uso.

Variável é um tipo particular de **identificador**.

Pode-se declarar e atribuir um valor inicial a uma variável:

Pode-se declarar mais de uma variável de mesmo tipo em uma única declaração.

Regras de nomenclatura: mesmas do Pascal.

A **declaração aqui é também uma definição** → **reserva espaço** para a variável na **memória**.

Valores são atribuídos às variáveis **com o operador “=”**;

Declarações e tipos primitivos de dados

■ Escopo de variável

❖ Define quando e onde uma variável pode ser utilizada.

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

int global = 10;

void f( int x ){
    int local = 10;
    printf( "Em f(): global= %d e local=%d\n",
            global, local );
    printf( "Parâmetro de f(): x= %d\n", x);
}

int main() {
    int local = 3;
    printf( "Em main(): global= %d e local=%d\n",
            global, local );
    f(5);
    {
        int local = -1;
        printf("Dentro de bloco: %d\n", local);
    }
    printf("E aqui?: %d\n", local);
    return 0;
}
```

Variáveis globais são definidas fora de qualquer função e existem durante toda a vida do programa.

Variáveis locais são definidas como como parâmetros e internamente às funções e existem durante o tempo de execução das funções.

Uma **variável definida** dentro de qualquer bloco ({ e }) existe somente dentro deste bloco.

Declarações e tipos primitivos de dados

■ Declarações e atribuições de variáveis

❖ Palavras reservadas de C (não usar como identificador!)

<code>auto</code>	<code>enum</code>	<code>restrict</code>	<code>unsigned</code>
<code>break</code>	<code>extern</code>	<code>return</code>	<code>void</code>
<code>case</code>	<code>float</code>	<code>short</code>	<code>volatile</code>
<code>char</code>	<code>for</code>	<code>signed</code>	<code>while</code>
<code>const</code>	<code>goto</code>	<code>sizeof</code>	<code>_Bool</code>
<code>continue</code>	<code>if</code>	<code>static</code>	<code>_Complex</code>
<code>default</code>	<code>inline</code>	<code>struct</code>	<code>_Imaginary</code>
<code>do</code>	<code>int</code>	<code>switch</code>	
<code>double</code>	<code>long</code>	<code>typedef</code>	
<code>else</code>	<code>register</code>	<code>union</code>	

Declarações e tipos primitivos de dados

■ Tipos primitivos de dados

Tipo	Explicação
char	Menor unidade endereçável da máquina. Consome 8 bits = 1 byte de memória. Consegue representar todo conjunto de caracteres básicos (ASCII) . É um tipo inteiro e pode ter ou não sinal. No GNU GCC (nosso compilador) seu intervalo numérico vai de -128 à 127 .
int	Tipo inteiro básico . Seu tamanho depende do compilador , mas consome no mínimo 16 bits = 2 bytes de memória. No GNU GCC ocupa 32 bits seu intervalo numérico vai de -2147483648 à 2147483647 .
long	Tipo inteiro longo . Seu tamanho depende do compilador , mas consome no mínimo 32 bits = 4 bytes de memória. No GNU GCC ocupa 32 bits seu intervalo numérico vai de -2147483648 à 2147483647 .
float	Tipo real com ponto flutuante de precisão simples . Formato interno definido pelo padrão IEEE 754 . Menor valor no GNU GCC 64 bits : 17549435082228750797e-38 . Maior valor : 3.40282346638528859812e+38 . Possui 24 dígitos de precisão .
double	Tipo real com ponto flutuante de precisão dupla . Formato interno definido pelo padrão IEEE 754 . Menor valor no GNU GCC 64 bits : 2.22507385850720138309e-308 . Maior valor : 1.79769313486231570815e+308 . Possui 53 dígitos de precisão .
void	Indica que nenhum valor está disponível. Utilizado na lista de parâmetros de funções que não tem parâmetros e quando uma função não possui valor de retorno (um procedimento, portanto). Usado também em ponteiros (assunto de outras aulas).

■ Literais

❖ Literal = símbolo de valor fixo

Literais inteiros	200 (decimal), 0310 (200 em octal), 0xC8 (200 em hexadecimal)
Literais em ponto flutuante	3.14159, 6.02E23
Literais de caractere	'a', '\\\ ' (caractere barra à esquerda), '\\n' (caractere de escape para nova linha), '\\\ ' (aspa simples)
Literais de cadeia de caracteres	"Computacao", "C:\\TEMP", "Ola mundo!\\n"

Declarações e tipos primitivos de dados

■ Definindo constantes

```
#include <stdio.h>
#define PI 3.14159
int main(void) {
    printf("%f\n", PI);
    return 0;
}
```

```
#include <stdio.h>
int main(void) {
    const float PI = 3.14159;
    printf("%f\n", PI);
    return 0;
}
```

Uma constante definida em `#define` é um símbolo que no programa é substituído antes da compilação. Assim, `PI` torna-se apenas o valor `3.14159`.

Não se usa “=” em `#define` !

Uma constante definida por `const` é um valor armazenado em uma posição de memória (como em variáveis), mas que não pode ser alterado. Vantagem: pode ser visto durante a depuração.

Expressões e operadores

■ Operador de atribuição (=)

- ❖ Armazena um determinado valor em uma variável;

- ❖ **Forma geral** em C: `variavel = expressao;`

❖ Conceitos de valor esquerdo e direito

- ✱ **Valor esquerdo (*lvalue*)**: designa um objeto (qualquer “coisa” que tenha um endereço na memória). Pode aparecer tanto do lado esquerdo de uma expressão de atribuição quanto do lado direito;

- ✱ **Valor direito (*rvalue*)**: é apenas um valor, sem endereço na memória. É uma expressão que pode aparecer do lado direito de uma expressão de atribuição, mas nunca do lado esquerdo.

- ✧ Constantes definidas com `const` são *lvalues*, mas não podem ser alteradas;

- ✧ Constantes definidas com `#define` são *rvalues* e não podem ser alteradas.

■ Operadores aritméticos

Operador	Significado	Exemplo
+	adição de dois valores	$z = x + y$
-	subtração de dois valores	$z = x - y$
*	multiplicação de dois valores	$z = x * y$
/	quociente de dois valores	$z = x / y$
%	resto de uma divisão	$z = x \% y$

❖ Lembre-se do programa a seguir ...

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    float x;
    x = 5 / 4;
    printf("x = %f\n", x);
    x = 5 / 4.0;
    printf("x = %f\n", x);
    return 0;
}
```


■ Operadores relacionais

Operador	Significado	Exemplo
>	Maior do que	x > 5
>=	Maior ou igual a	x >= 10
<	Menor do que	x < 5
<=	Menor ou igual a	x <= 10
==	Igual a	x == 0
!=	Diferente de	x != 0

❖ **Nota:** em C, uma **expressão falsa** possui o **valor zero**. Para qualquer outro resultado diferente de zero, a expressão será considerada **verdadeira**.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("%d\n", 2>3 );
    if(3.14)
        printf("Verdadeiro!\n");
    return 0;
}
```

■ Operadores lógicos

Operador	Significado	Exemplo
&&	Operador E	(x >= 0 && x <= 9)
	Operador OU	(a == 'F' b != 32)
!	Operador NEGAÇÃO	! (x == 10)

❖ Regras básicas

✱ $x \&\& y$ é **verdadeiro** apenas quando x e y são ambos **verdadeiros**;

✱ $x || y$ é **falso** apenas quando x e y são **falsos**;

✱ $!x$ **inverte** o sentido lógico de x : se x é **verdadeiro**, $!x$ resulta **falso** e; se x é **falso**, $!x$ resulta **verdadeiro**.

■ Operadores de atribuição abreviados

Operador	Significado	Exemplo		
<code>+=</code>	soma e atribui	<code>x += y</code>	igual	<code>x = x + y</code>
<code>-=</code>	subtrai e atribui	<code>x -= y</code>	igual	<code>x = x - y</code>
<code>*=</code>	multiplica e atribui	<code>x *= y</code>	igual	<code>x = x * y</code>
<code>/=</code>	divide e atribui quociente	<code>x /= y</code>	igual	<code>x = x / y</code>
<code>%=</code>	divide e atribui resto	<code>x %= y</code>	igual	<code>x = x % y</code>

■ Operadores de incremento

Operador	Significado	Exemplo	Resultado
++	incremento	++x ou x++	$x = x + 1$
--	decremento	--x ou x--	$x = x - 1$

❖ Observações importantes:

- **++x (pré-incremento):** soma 1 à variável **x** antes de utilizar seu valor.
- **x++ (pós-incremento):** soma 1 à variável **x** depois de utilizar seu valor.
- **--x (pré-decremento):** subtrai 1 da variável **x** antes de utilizar seu valor.
- **x-- (pós-decremento):** subtrai 1 da variável **x** depois de utilizar seu valor.

■ Operador condicional ternário

❖ Verifica o valor lógico de uma expressão e retorna o valor indicado na primeira parte se ela for verdadeira ou da segunda parte se ela for falsa:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i = 10, j = 20;
    int maior = ( i > j ) ? i : j;
    printf( "Maior: %d\n", maior );
    return 0;
}
```

■ Operador “,”

❖ Além de pontuador (separar parâmetros nas funções) o operador “,” permite o encadeamento de expressões, resultando o valor da última.

❖ Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int x, y = 3;
    x = ( y++, y += 3 );
    printf( "Valor de x = %d\n", x );
    printf( "Valor de x = %d\n", x );
    return 0;
}
```

■ Operador `sizeof`

❖ Retorna o número de bytes ocupados por uma variável ou nome de tipo.

❖ Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i = 10;
    float f = -1.234;
    double d = 34985.988;
    char c = 'A';
    printf("i ocupa: %d bytes\n", sizeof(i));
    printf("f ocupa: %d bytes\n", sizeof(f));
    printf("d ocupa: %d bytes\n", sizeof(d));
    printf("c ocupa: %d bytes\n", sizeof(c));
    printf("long ocupa: %d bytes\n", sizeof(long));
    return 0;
}
```

■ Conversões de tipos

❖ A conversão de tipo (*type cast*) é uma forma explícita para converter um tipo em outro;

❖ Forma: (nome_do_tipo) expressao

❖ Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    float x;
    x = (float) 5 / 4;
    printf("x = %f\n", x);
    x = 5 / 4.0;
    printf("x = %f\n", x);
    return 0;
}
```


Expressões e operadores

■ Precedência e associatividade dos operadores

Precedence	Operators	Associativity
1.	Postfix operators: [] () . -> ++ -- (<i>type name</i>){ <i>list</i> }	Left to right
2.	Unary operators: ++ -- ! ~ + - * & sizeof	Right to left
3.	The cast operator: (<i>type name</i>)	Right to left
4.	Multiplicative operators: * / %	Left to right
5.	Additive operators: + -	Left to right
6.	Shift operators: << >>	Left to right
7.	Relational operators: < <= > >=	Left to right
8.	Equality operators: == !=	Left to right
9.	Bitwise AND: &	Left to right
10.	Bitwise exclusive OR: ^	Left to right
11.	Bitwise OR:	Left to right
12.	Logical AND: &&	Left to right
13.	Logical OR:	Left to right
14.	The conditional operator: ? :	Right to left
15.	Assignment operators: = += -= *= /= %= &= ^= = <<= >>=	Right to left
16.	The comma operator: ,	Left to right

■ Saída de dados com `printf`

- ❖ Exibe na tela dados formatados;
- ❖ A função `printf` possui um número variável de parâmetros onde o primeiro é sempre uma cadeia de caracteres;
- ❖ Se houver mais de um parâmetro passado a `printf`, estes deverão ser passados após a cadeia de caracteres em questão e nela indicados por especificadores de formação apropriados.

Alguns formatadores de saída	
<code>%c</code>	Escreve um caractere (char)
<code>%d</code> ou <code>%i</code>	Escreve um número inteiro (int ou char)
<code>%f</code>	Escreve um número real (float ou double)
<code>%e</code> ou <code>%E</code>	Escreve um número em notação científica
<code>%s</code>	Escreve uma string (char [])

■ Entrada de dados com `scanf`

- ❖ Lê dados formatados a partir do teclado;
- ❖ A função `scanf` possui um número variável de parâmetros onde o primeiro é sempre uma cadeia de caracteres;
- ❖ Se houver mais de um parâmetro passado a `scanf`, estes deverão ser passados após a cadeia de caracteres em questão e nela indicados por especificadores de formação apropriados – os demais parâmetros são variáveis passadas por referência (&).

Alguns formatadores de entrada	
<code>%c</code>	leitura de um caractere (char)
<code>%d</code> ou <code>%i</code>	leitura de número inteiro (int ou char)
<code>%f</code>	leitura de número real (float ou double)

Comandos para controle de fluxo de programa

■ Comando `if`

❖ Formas

```
if(expr_logica) {  
    comando1;  
    comando2;  
}
```

```
if(expr_logica) {  
    comando1;  
    comando2;  
}  
else {  
    comando3;  
    comando4;  
}
```

```
if(expr_logica1) {  
    comando1;  
    comando2;  
}  
else if(expr_logica2) {  
    comando3;  
    comando4;  
}  
else {  
    comando5;  
    comando6;  
}
```

❖ Exemplo

```
if( delta < 0 )  
    printf("Nao existem raizes reais \n");  
else {  
    r1 = (-b + sqrt(delta))/(2*a);  
    r2 = (-b - sqrt(delta))/(2*a);  
    if( delta == 0 )  
        printf("Existe uma unica raiz: %4.2f\n", r1);  
    else  
        printf("As raizes são: %4.2f e %4.2f\n", r1, r2);  
}
```

Comandos para controle de fluxo de programa

■ Comando switch

❖ Forma

```
switch (expr_inteira) {  
    case valor1:  
        comando1;  
        comando2;  
        break;  
    case valor2:  
        comando3;  
        break;  
    default:  
        comando4;  
        break;  
}
```

❖ Exemplo

```
switch (toupper(c)) {  
    case 'A':  
    case 'E':  
    case 'I':  
    case 'O':  
    case 'U':  
        printf("%c eh uma vogal!\n", c);  
        break;  
    default:  
        printf("%c eh outro simbolo\n!", c);  
        break;  
}
```

Comandos para controle de fluxo de programa

■ Comando `for`

❖ Forma

```
for( expr_inicial; cond_repet; expr_repet){  
    comando1;  
    comando2;  
}
```

❖ Exemplo

```
for( i=0; i<N; i++)  
{  
    printf("Digite o valor %d: ", i + 1);  
    scanf("%f", &valores[i]);  
    soma = soma + valores[i];  
}
```

Comandos para controle de fluxo de programa

■ Comando `while`

❖ Forma

```
while ( expr )  
{  
    comando1;  
    comando2;  
}
```

❖ Exemplo

```
while ( r1 != r2 )  
    if (r1 > r2)  
        r1 = r1 - r2;  
    else  
        r2 = r2 - r1;
```

Comandos para controle de fluxo de programa

■ Comando do-while

❖ Forma

```
do {  
    comando1;  
    comando2;  
} while ( expr );
```

❖ Exemplo

```
do {  
    digito_direito = numero % 10;  
    printf("%i", digito_direito);  
    numero = numero / 10;  
} while ( numero != 0 );
```


Bibliografia

- DEITEL, H. M. **C: Como programar**. Trad. de João Eduardo Nóbrega Tortello; rev. téc. Alvaro Rodrigues Antunes. São Paulo, SP: Pearson Education, 2003. 1153 p. ISBN 8534614598.
- SCHILDT, Herbert. **C Completo e total**. [Título original: C: the complete reference]. Trad. e rev. téc. Roberto Carlos Mayer. 3. ed. São Paulo, SP: Pearson Education do Brasil, 2011. 827 p. ISBN 9788534605953.