

Começando a Codificar

variáveis, operadores, expressões, entrada e saída

```
/*  
=====
```

Name	:	hello.c
Author	:	lincoln
Version	:	
Copyright	:	Your copyright notice
Description	:	Hello World in C, Ansi-style

```
=====
```

```
*/  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void)  
{  
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */  
    return 0;  
}
```

```
/*
```

```
=====
Name       : hello.c
Author      : lincoln
Version     :
Copyright   : Your copyright notice
Description : Hello World in C, Ansi-style
=====
```

```
*/
```

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

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

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

```
}
```

O que essa
imagem ilustra?
O que é isso?

```
/*  
=====   
Name      : hello.c  
Author    : lincoln  
Version   :  
Copyright : Your copyright notice  
Description : Hello World in C, Ansi-style  
=====   
*/
```

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

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

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

Esse programa apenas
exibe essa mensagem
na tela.

O que essa
imagem ilustra?
O que é isso?

Isso é um código-fonte em C!
A partir de agora vai ficar
mais fácil para você
identificar um código quando
ver um, já que eles, assim
como as cartas, podem
apresentar uma estrutura bem
definida.

```
/*  
=====   
Name       : hello.c  
Author     : lincoln  
Version    :  
Copyright  : Your copyright notice  
Description : Hello World in C, Ansi-style  
=====   
*/
```

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

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

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

```
}
```

um texto de apresentação (comentários)



```
/*  
=====   
Name       : hello.c  
Author      : lincoln  
Version     :  
Copyright   : Your copyright notice  
Description : Hello World in C, Ansi-style  
=====   
*/
```

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

um texto de apresentação (comentários)

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

comentários são ignorados pelo computador (só servem pra gente)

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

```
}
```

Tudo que for escrito entre um `/*` e um `*/` é comentário. Todo texto após um `//` também. Comentários não são instruções e são ignorados pelo compilador.

Os comentários só servem então para outros programadores lendo o código. Nesse contexto, eles podem ser utilizados para inserir informações pertinentes sobre o programa, instrução ou um trecho do código.

```
/*  
=====   
Name      : hello.c  
Author    : lincoln  
Version   :  
Copyright : Your copyright notice  
Description : Hello World in C, Ansi  
=====   
*/
```

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

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

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

```
}
```

um texto de apresentação (comentários)

comentários são ignorados pelo computador (só servem pra gente)

```
/*
```

```
=====
Name       : hello.c
Author      : lincoln
Version     :
Copyright   : Your copyright notice
Description : Hello World in C, Ansi-style
=====
```

```
*/
```

informa as bibliotecas que serão utilizadas (o conjunto de instruções que poderão ser utilizadas)

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

```
int main(void)
```

```
{
```

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

```
}
```



```
/*  
=====   
Name      : hello.c  
Author    : lincoln  
Version   :  
Copyright : Your copyright notice  
Description : Hello World in C, Ansi-style  
=====   
*/
```

informa as bibliotecas que serão utilizadas (o conjunto de instruções que poderão ser utilizadas)

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

```
int main(void)  
{  
    puts("!!!Hello World!!!"); /* imprime !!!  
    return 0;  
}
```

É como se cada biblioteca fosse um dicionário que define o vocabulário de instruções que poderemos utilizar no nosso código.

Por exemplo, a biblioteca matemática **math.h** permite que a utilização de instruções de raiz quadrada, potenciação, seno e outras.

```
/*
```

```
=====
Name       : hello.c
Author     : lincoln
Version    :
Copyright  : Your copyright notice
Description: Hello World in C, Ansi-style
=====
```

```
*/
```

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

```
int main(void)
{
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
    return 0;
}
```

definição da função *main*, o ponto de partida do programa



```
/*
```

```
=====
Name       : hello.c
Author     : lincoln
Version    :
Copyright  : Your copyright notice
Description: Hello World in C, Ansi-style
=====
```

```
*/
```

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

```
int main(void)
{
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!
    return 0;
}
```

definição da função *main*, o ponto de partida do programa

Todo programa em C a ser executado em um sistema operacional deve apresentar essa função *main*.

```
/*
```

```
=====
Name       : hello.c
Author     : lincoln
Version    :
Copyright  : Your copyright notice
Description: Hello World in C, Ansi-style
=====
```

```
*/
```

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

```
int main(void)
```

```
{
```

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;
```

```
}
```

instruções (comandos)

```
/*  
=====   
Name      : hello.c  
Author    : lincoln  
Version   :  
Copyright : Your copyright notice  
Description : Hello World in C, Ansi-style  
=====   
*/
```

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

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

```
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */
```

```
    return 0;  
}
```

instruções (comandos)

Em C, todas as instruções devem aparecer sempre dentro de alguma função. Nossos programas iniciais serão compostos por uma sequência de instruções na função *main*.

```

/*
=====
Name      : hello.c
Author    : lincoln
Version   :
Copyright : Your copyright notice
Description : Hello World in C, Ansi-style
=====
*/

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

int main(void)
{
    puts("!!!Hello World!!!"); /* imprime !!!Hello World!!! */

    return 0;
}

```

Agora é um bom momento para aprender como compilar e executar esse código-fonte. Faça a atividade **Alô Mundo** do Sala de Aula. [Link do vídeo](#).

Pausa para a atividade *Alô Mundo*

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

E esse código-fonte aqui? Que programa você acha que ele implementa?


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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Bom, algumas palavras, valores e expressões parecem indicar que ocorre o cálculo da área de um círculo.

E esse código-fonte aqui? Que programa você acha que ele implementa?

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

As instruções são as linhas inseridas *dentro* da função **main**.

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

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

Essas duas linhas/instruções indicam, cada uma, que o programa precisa de *variáveis*, ou seja, que o programa precisa armazenar e manipular dois valores na memória. Os nomes das variáveis são **raio** e **area**.

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Já vimos uma instrução bem parecida com essa no exemplo **Alô Mundo**. Portanto, já sabemos que ela fará a mensagem especificada aparecer na tela.

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

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

```
int main(void) {
    int raio;
    float area;
```

```
    printf("Bem-vindo,\n");
```

```
    raio = 5;
    area = 3.14 * raio*raio;
```

```
    printf("Para r = %d, a = %f\n", raio, area);
```

```
    return 0;
```

```
}
```

Aqui estamos atribuindo o valor **5** à variável **raio**. Acredito que era fácil deduzir, não era? Parece muito com uma linha que poderíamos usar em um exercício de matemática.

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;
    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Aqui estamos atribuindo o valor da expressão ao lado direito da igualdade à variável **area**. Como na linha anterior, parece muito com o que estamos acostumados na matemática. Além disso, com algum esforço, conseguimos identificar que a expressão é πr^2 .

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Mais uma vez, sempre que começar com **printf** já sabemos que alguma mensagem será exibida na tela. Mas está bem mais complicado entender qual mensagem será exibida nesse caso.

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.


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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;
```

```
    printf("Para r = %d, a = %f\n", raio, area);
```

```
    return 0;
}
```

Essa instrução especial serve para encerrar a execução do programa. O Valor zero serve para indicar um encerramento sem erros.

Ainda vamos aprender com detalhes, mas vamos linha a linha identificar o que cada instrução faz.

[Veja o vídeo](#) com a compilação e execução deste exemplo.

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

inclusão biblioteca 1

inclusão biblioteca 2

...

inclusão biblioteca n

outras definições

outras definições

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

instrução 1

instrução 2

instrução 3

instrução 4

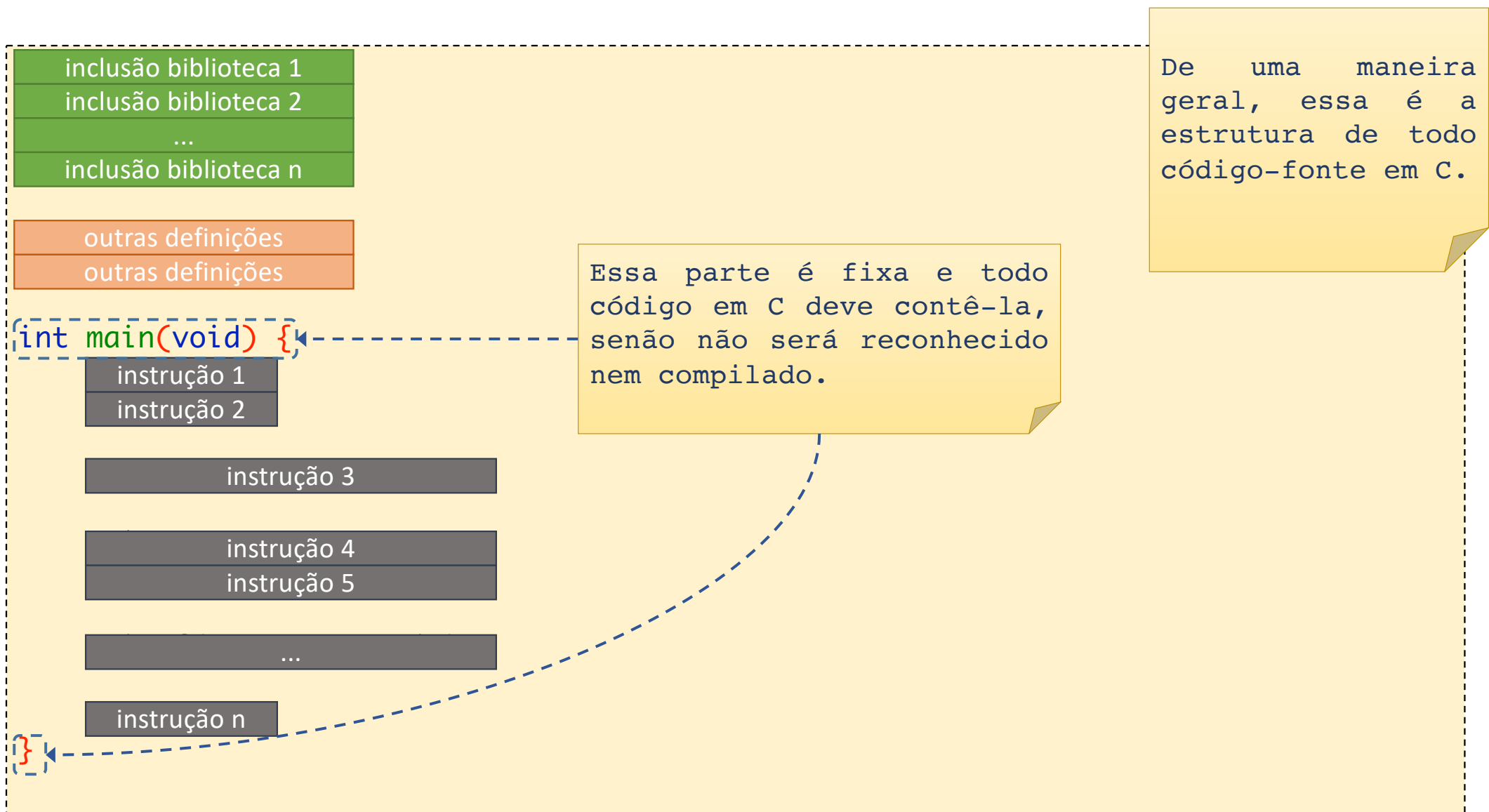
instrução 5

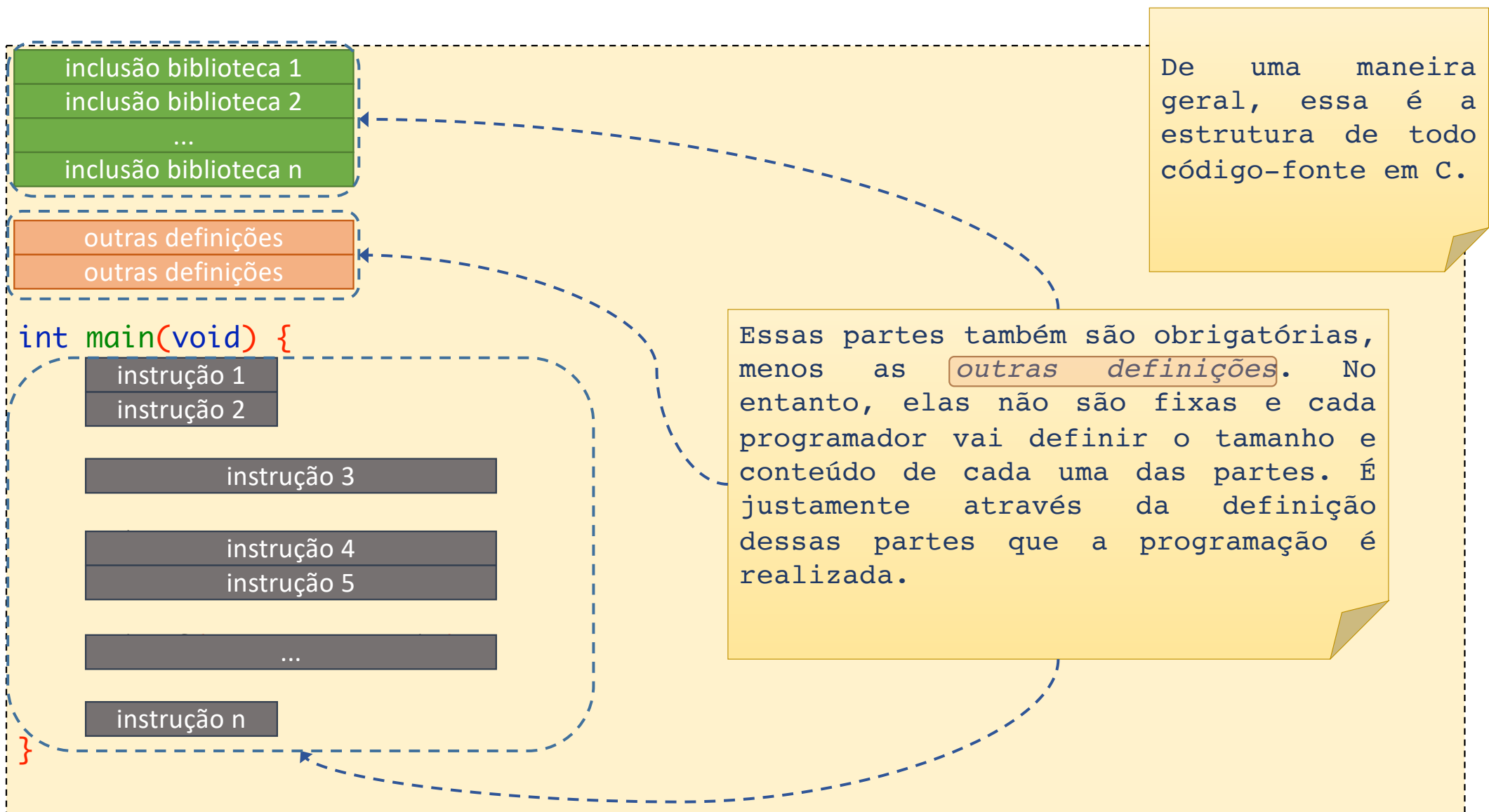
...

instrução n

```
}
```

De uma maneira geral, essa é a estrutura de todo código-fonte em C.





inclusão biblioteca 1
inclusão biblioteca 2
...
inclusão biblioteca n

outras definições
outras definições

Como já vimos superficialmente,
aqui indicamos as bibliotecas
utilizadas pelo programa.

De uma maneira
geral, essa é a
estrutura de todo
código-fonte em C.

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

instrução 1

instrução 2

instrução 3

instrução 4

instrução 5

...

instrução n

```
}
```

inclusão biblioteca 1
inclusão biblioteca 2
...
inclusão biblioteca n

outras definições
outras definições

Nessa parte opcional
faremos a definição de
variáveis globais, funções,
tipos, alusões, enumerações
e outras coisas.

De uma maneira
geral, essa é a
estrutura de todo
código-fonte em C.

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

instrução 1

instrução 2

instrução 3

instrução 4

instrução 5

...

instrução n

```
}
```

inclusão biblioteca 1
inclusão biblioteca 2
...
inclusão biblioteca n

outras definições
outras definições

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

instrução 1
instrução 2

instrução 3

instrução 4
instrução 5

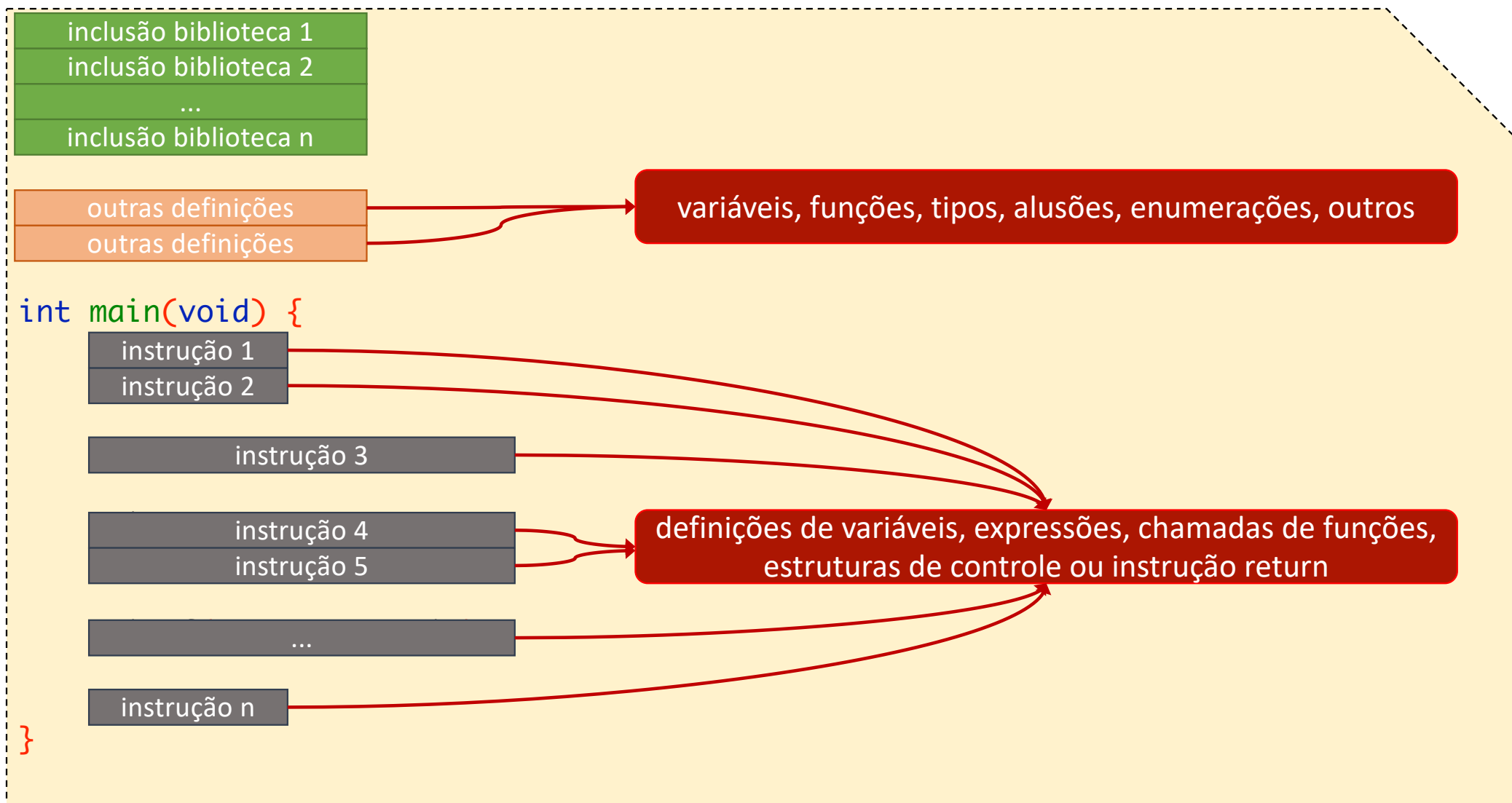
...

instrução n

```
}
```

De uma maneira geral, essa é a estrutura de todo código-fonte em C.

E na função **main** é que definimos o funcionamento do programa, que programamos a máquina. Cada instrução pode ser uma dentre: definições de variáveis, expressões, chamadas de funções, estruturas de controle ou instrução return.



Como está organizado esse código-fonte em relação à sua estrutura?

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Como está organizado esse código-fonte em relação à sua estrutura?

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

inclusão das bibliotecas utilizadas

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

```
    int raio;
    float area;
```

definição de variáveis

```
    printf("Bem-vindo,\n");
```

```
    raio = 5;
    area = 3.14 * raio*raio;
```

expressões

```
    printf("Para r = %d, a = %f\n", raio, area);
```

chamadas de funções

```
    return 0;
```

instrução especial

```
}
```

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

```
int main(void) {
    int raio;
    float area;

    printf("Bem-vindo,\n");

    raio = 5;
    area = 3.14 * raio*raio;

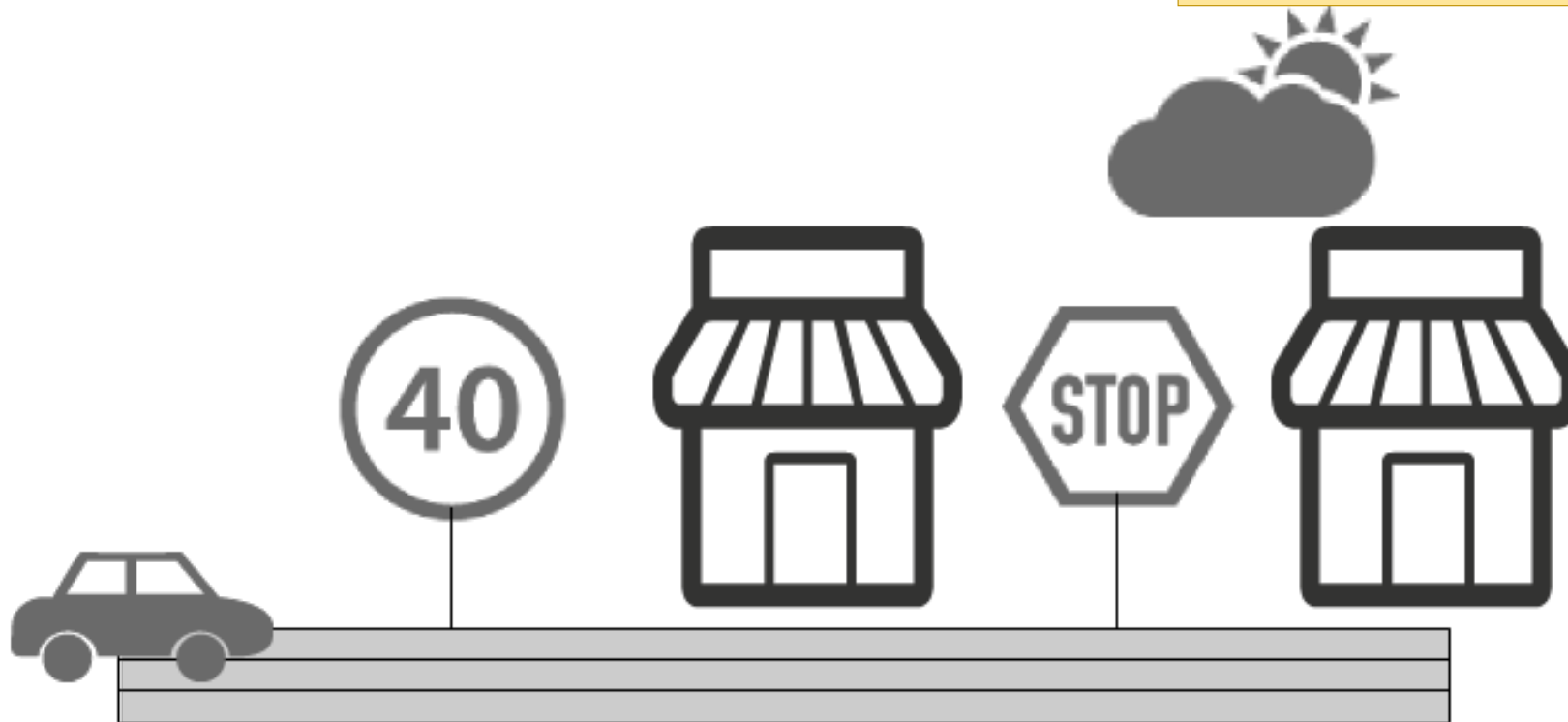
    printf("Para r = %d, a = %f\n", raio, area);

    return 0;
}
```

Já vimos o computador executar esse programa compilado. Mas é importante compreender qual a regra seguida pelo computador para executar um programa.

Fluxo de Execução

Assim como uma via sinalizada traz instruções para um motorista, podemos pensar um programa como uma via com instruções para o computador.



Fluxo de Execução

Assim como uma via sinalizada traz instruções para um motorista, podemos pensar um programa como uma via com instruções para o computador.

O computador é o motorista/automóvel e as sinalizações são as instruções.



Fluxo de Execução

Assim como uma via sinalizada traz instruções para um motorista, podemos pensar um programa como uma via com instruções para o computador.

O computador é o motorista/automóvel e as sinalizações são as instruções.



Assim como um motorista seguindo uma via e respeitando a sinalização encontrada, o computador executa o programa executando cada instrução uma única vez e na ordem em que aparecem. Isso é o **Fluxo de Execução Natural** do programa.

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

```
int main(void) {
    int raio; ①
    float area; ②

    printf("Bem-vindo,\n"); ③

    raio = 5; ④
    area = 3.14 * raio*raio; ⑤

    printf("Para r = %d, a = %f\n", raio, area); ⑥

    return 0; ⑦
}
```

Seguindo o fluxo de execução natural, o programa do quadro é sempre executado na ordem mostrada.

Quais são os passos do algoritmo necessários para implementar um programa que vai nos dizer quantas latas de tintas são necessárias para pintar uma parede?



Quais são os passos do algoritmo necessários para implementar um programa que vai nos dizer quantas latas de tintas são necessárias para pintar uma parede?

Passos do algoritmo:

- Perguntar as dimensões da parede.
- Perguntar o rendimento da lata de tinta.
- Realizar o cálculo.
- Informar o resultado.



Quais são os passos do algoritmo necessários para implementar um programa que vai nos dizer quantas latas de tintas são necessárias para pintar uma parede?

E quais seriam os tipos de instruções na linguagem de programação necessárias para implementar esses passos?

Passos do algoritmo:

- Perguntar as dimensões da parede.
- Perguntar o rendimento da lata de tinta.
- Realizar o cálculo.
- Informar o resultado.



Quais são os passos do algoritmo necessários para implementar um programa que vai nos dizer quantas latas de tintas são necessárias para pintar uma parede?

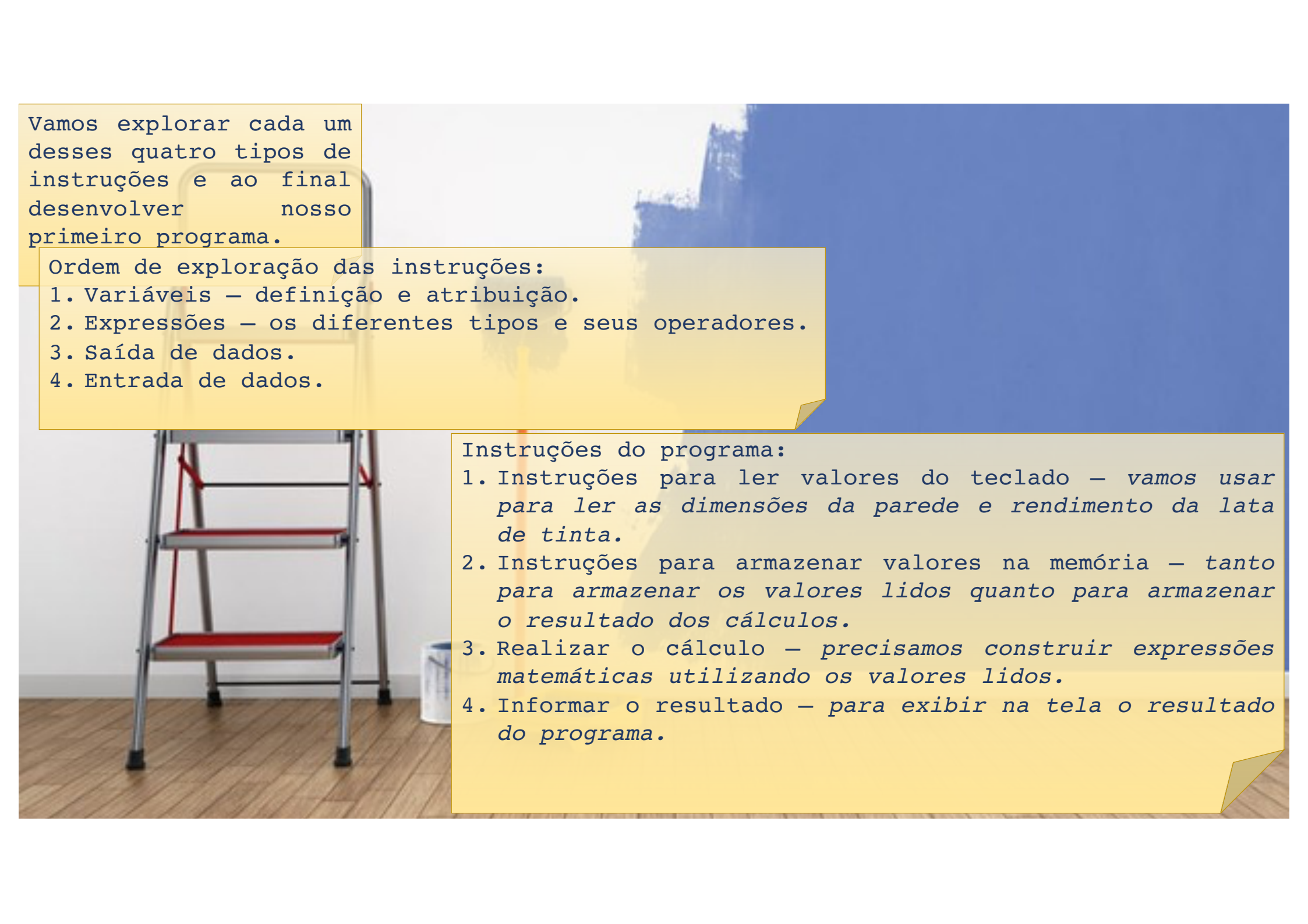
E quais seriam os tipos de instruções na linguagem de programação necessárias para implementar esses passos?

Passos do algoritmo:

- Perguntar as dimensões da parede.
- Perguntar o rendimento da lata de tinta.
- Realizar o cálculo.
- Informar o resultado.

Instruções do programa:

1. Instruções para ler valores do teclado – vamos usar para ler as dimensões da parede e rendimento da lata de tinta.
2. Instruções para armazenar valores na memória – tanto para armazenar os valores lidos quanto para armazenar o resultado dos cálculos.
3. Realizar o cálculo – precisamos construir expressões matemáticas utilizando os valores lidos.
4. Informar o resultado – para exibir na tela o resultado do programa.



Vamos explorar cada um desses quatro tipos de instruções e ao final desenvolver nosso primeiro programa.

Ordem de exploração das instruções:

1. Variáveis – definição e atribuição.
2. Expressões – os diferentes tipos e seus operadores.
3. Saída de dados.
4. Entrada de dados.

Instruções do programa:

1. Instruções para ler valores do teclado – vamos usar para ler as dimensões da parede e rendimento da lata de tinta.
2. Instruções para armazenar valores na memória – tanto para armazenar os valores lidos quanto para armazenar o resultado dos cálculos.
3. Realizar o cálculo – precisamos construir expressões matemáticas utilizando os valores lidos.
4. Informar o resultado – para exibir na tela o resultado do programa.

Variáveis

Imagine a memória do computador como a dispensa de uma casa. Quando a dispensa está vazia suas prateleiras estão livres e não armazenam nenhum mantimento - ou seja, elas estão com toda a capacidade disponível.

Variáveis

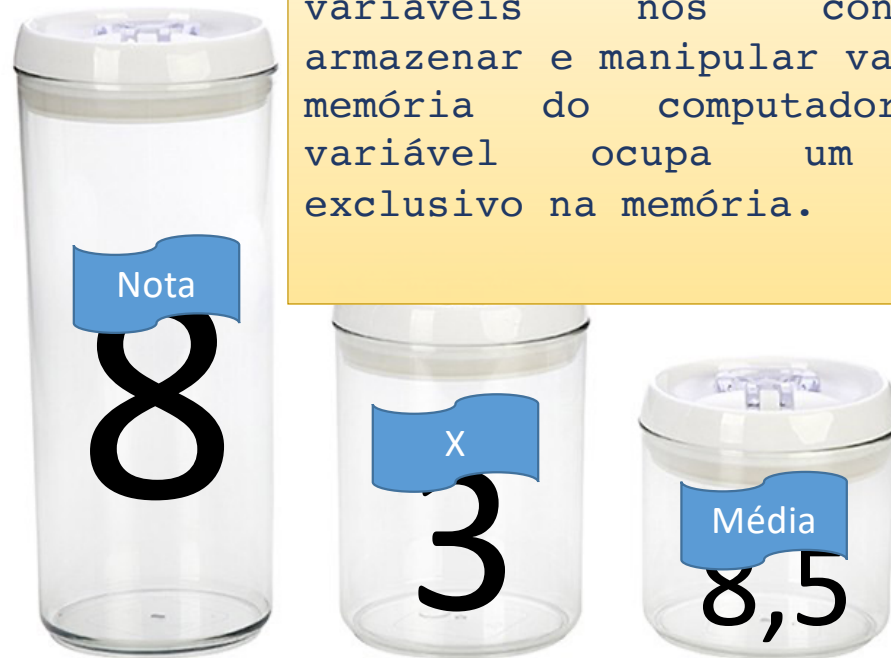
Imagine a memória do computador como a dispensa de uma casa. Quando a dispensa está vazia suas prateleiras estão livres e não armazenam nenhum mantimento - ou seja, elas estão com toda a capacidade disponível.

Uma dispensa cheia se parece mais como essa da imagem, com seus espaços ocupados com recipientes para os mantimentos.



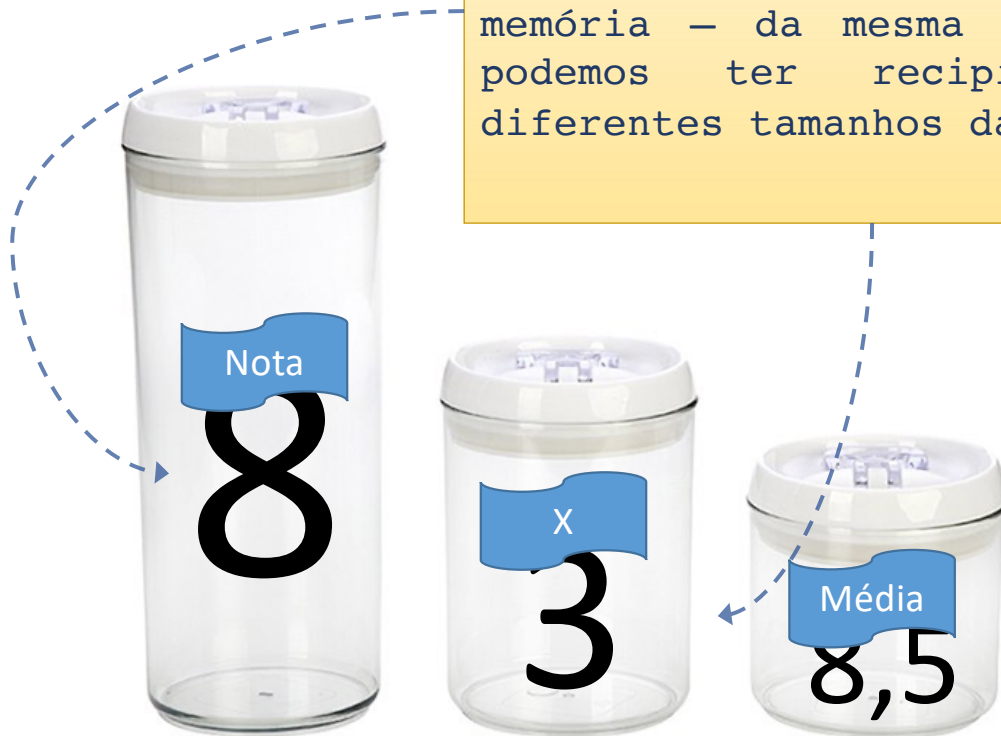
Variáveis

Nessa analogia, as variáveis são como os recipientes. Através das variáveis nós conseguimos armazenar e manipular valores na memória do computador. Cada variável ocupa um espaço exclusivo na memória.



Variáveis

Toda variável armazena um **valor**.
Dependendo do tipo de valor armazenado, as variáveis podem ocupar mais ou menos espaço da memória — da mesma forma como podemos ter recipientes de diferentes tamanhos da dispensa.



Variáveis



Variáveis



O nome de uma variável funciona como um rótulo para o dado armazenado.

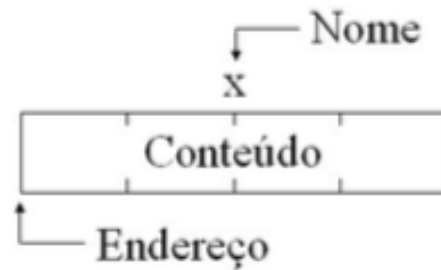
Em um código-fonte, o nome da variável representa o seu conteúdo. Ou seja, utilizamos o nome da variável para consultar ou definir o valor armazenado na memória.



Variáveis

Sendo mais formal:

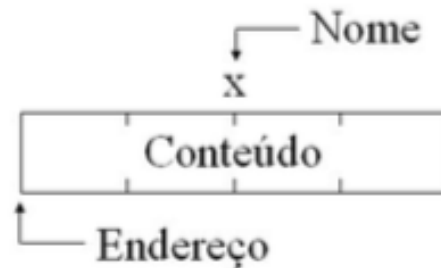
Uma variável representa por meio de um nome o conteúdo de um espaço contínuo de memória.



Variáveis

Sendo mais formal:

Uma variável representa por meio de um nome o conteúdo de um espaço contínuo de memória.



Cada variável também possui um **endereço**, que é a posição na memória onde foi armazenada.

Como tudo é armazenado na memória como dados binários, os *bits*, também devemos associar cada variável a um **tipo de dado**, indicando como esses *bits* devem ser interpretados no contexto do programa.

Variáveis e Atribuições

Agora vamos pensar na memória do computador como uma mesa contendo organizadores vazios e aprender o código necessário para definir e manipular as variáveis.



Variáveis e Atribuições

```
int i;  
float x;
```



Variáveis e Atribuições

```
int i;  
float x;
```

Assim definimos uma variável.
Em uma linha, começamos
especificando um tipo de dado
e separamos o nome desejado
para a variável com um espaço.



Variáveis e Atribuições

```
int i;  
float x;
```

Assim definimos uma variável. Em uma linha, começamos especificando um tipo de dado e separamos o nome desejado para a variável com um espaço.

Por enquanto vamos usar os tipos:

- **int** – para manipular números inteiros.
- **float** – para manipular números reais.
- **char** – para manipular caracteres (letras/dígitos).



#01



#02



#03



#04



#05

Variáveis e Atribuições

```
int i;  
float x;
```

Definições de variáveis associam um espaço da memória ao nome da variável.



Variáveis e Atribuições

```
int i;  
float x;
```

Definições de variáveis associam um espaço da memória ao nome da variável.



Variáveis e Atribuições

```
int i;  
float x;  
  
i = 1;  
x = 2.3;
```



Variáveis e Atribuições

```
int i;  
float x;
```

```
i = 1;  
x = 2.3;
```

As instruções de atribuição definem o valor da variável — do espaço dela na memória — a partir daquele ponto do programa.



Variáveis e Atribuições

```
int i;  
float x;
```

```
i = 1;  
x = 2.3;
```

As instruções de atribuição definem o valor da variável – do espaço dela na memória – a partir daquele ponto do programa.

Devemos atribuir valores que correspondam ao tipo da variável.



Variáveis e Atribuições

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;  
  
umInteiro  = 7;  
desconto   = 1.20 + 0.50;  
preco      = 5.50 - desconto;  
minhaLetra = 'K';
```

Variáveis e Atribuições

Sempre devemos definir as variáveis antes de sua utilização. É uma boa prática iniciar o código com a definição de todas as variáveis.

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;
```

```
umInteiro = 7;  
desconto  = 1.20 + 0.50;  
preco     = 5.50 - desconto;  
minhaLetra = 'K';
```


Variáveis e Atribuições

```
int umInteiro;  
float preco, desconto;  
char minhaLetra;
```

```
umInteiro = 7;  
desconto = 1.20 + 0.50;  
preco = 5.50 - desconto;  
minhaLetra = 'K';
```

Escolha sempre nomes significativos para suas variáveis, que representem bem o significado do valor para o programa. Evite nomes como a, b, c, x, ou y.

Também defina tipos adequados. Se o valor armazenado for um número inteiro, defina como **int** e não **float**, mesmo que um número real possa armazenar um inteiro.

Variáveis e Atribuições

Para definir mais de uma variável com um mesmo tipo em uma única linha, basta separar os nomes por vírgula.

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;
```

```
umInteiro = 7;  
desconto  = 1.20 + 0.50;  
preco     = 5.50 - desconto;  
minhaLetra = 'K';
```

Variáveis e Atribuições

Vamos convencionar de escrever os nomes das variáveis com letras minúsculas e usar uma letra maiúscula ao início de cada palavra, caso o nome seja composto por várias palavras.

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;
```

```
umInteiro    = 7;  
desconto     = 1.20 + 0.50;  
preco        = 5.50 - desconto;  
minhaLetra   = 'K';
```

Variáveis e Atribuições

Os nomes das variáveis **sempre** aparecem ao lado esquerdo da atribuição.

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;  
  
umInteiro = 7;  
desconto  = 1.20 + 0.50;  
preco     = 5.50 - desconto;  
minhaLetra = 'K';
```

Variáveis e Atribuições

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;
```

```
umInteiro = 7;  
desconto  = 1.20 + 0.50;  
preco     = 5.50 - desconto;  
minhaLetra = 'K';
```

Ao lado direito da expressão contém um valor único ou uma expressão. Esse valor único ou o resultado da expressão é o que será atribuído à variável.

Variáveis e Atribuições

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;  
  
umInteiro = 7;  
desconto  = 1.20 + 0.50;  
preco     = 5.50 - desconto;  
minhaLetra = 'K';
```

Quando o nome de uma variável aparece em uma expressão, é considerado o valor que ela armazena no momento de execução da instrução para realização do cálculo.

Variáveis e Atribuições

```
int    umInteiro;  
float  preco, desconto;  
char   minhaLetra;  
  
umInteiro  = 7;  
desconto   = 1.20 + 0.50;  
preco      = 5.50 - desconto;  
minhaLetra = 'K';
```

Já deu para perceber que toda instrução em C deve terminar com um ; e ocupar uma única linha.

Variáveis e Atribuições

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

int main(void)
{
    int    umInteiro;
    float  preco, desconto;
    char   minhaLetra;

    umInteiro  = 7;
    desconto   = 1.20 + 0.50;
    preco      = 5.50 - desconto;
    minhaLetra = 'K';

    return 0;
}
```

Assim fica o código-fonte completo do exemplo pronto para compilação.

Atribuições e Expressões

```
int    umInteiro;  
char   umaLetra;  
float  nota1, nota2, media;  
  
umInteiro = 1;  
umaLetra = 'A';  
nota1 = 8.5;  
nota2 = nota1;  
media = (nota1 + nota2) / 2;  
  
umInteiro = umInteiro + 2;
```

Exercício: Quais serão os valores armazenados nas variáveis após a execução desse trecho de código?

```
umInteiro: ?  
umaLetra: ?  
nota1: ?  
nota2: ?  
media: ?
```

Atribuições e Expressões

```
int    umInteiro;  
char   umaLetra;  
float  nota1, nota2, media;  
  
umInteiro = 1;  
umaLetra = 'A';  
nota1 = 8.5;  
nota2 = nota1 + 1;  
media = (nota1 + nota2) / 2;  
  
umInteiro = umInteiro + 2;
```

Exercício: Quais serão os valores armazenados nas variáveis após a execução desse trecho de código?

```
umInteiro: 3  
umaLetra: A  
nota1: 8.5  
nota2: 9.5  
media: 9.0
```

Operadores e Expressões

- Expressões aritméticas
- Expressões relacionais
- Expressões lógicas

Esses são os três tipos de expressões que podemos construir em C. Eles são equivalentes aos tipos matemáticos e utilizam um conjunto semelhante de operadores.

Operadores e Expressões

- Expressões aritméticas
- Expressões relacionais
- Expressões lógicas

Esses são os três tipos de expressões que podemos construir em C. Eles são equivalentes aos tipos matemáticos e utilizam um conjunto semelhante de operadores.

As expressões aritméticas associam o uso dos operadores aritméticos (soma, subtração, etc) com valores numéricos. O resultado é também um valor numérico. Por exemplo, $2 + 3$ resulta em **5**.

Operadores e Expressões

- Expressões aritméticas
- Expressões relacionais
- Expressões lógicas

Esses são os três tipos de expressões que podemos construir em C. Eles são equivalentes aos tipos matemáticos e utilizam um conjunto semelhante de operadores.

As expressões relacionais associam o uso dos operadores relacionais (maior, igual, menor do que, etc) com valores numéricos. O resultado é valor booleano (**verdadeiro** ou **falso**). Por exemplo, **2 > 3** resulta em **falso**.

Operadores e Expressões

- Expressões aritméticas
- Expressões relacionais
- Expressões lógicas

Esses são os três tipos de expressões que podemos construir em C. Eles são equivalentes aos tipos matemáticos e utilizam um conjunto semelhante de operadores.

Já as expressões lógicas associam o uso dos operadores lógicos (**e**, **ou** e **não**) com valores booleanos (**verdadeiro** ou **falso**). O resultado é também um valor booleano. Por exemplo, **verdadeiro e falso** resulta em **falso**.

Propriedades dos Operadores

- Aridade
- Resultado
- Precedência
- Associatividade

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

Propriedades dos Operadores

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

- Aridade
- Resultado
- Precedência
- Associatividade

Aridade é a propriedade que define o número de operandos necessários para utilização do operador. Por exemplo, a soma tem aridade 2 pois precisamos de dois operandos, como em $5 + 7$.

Propriedades dos Operadores

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

- Aridade
- Resultado
- Precedência
- Associatividade

Resultado é o valor gerado pela aplicação do operador. Por exemplo, o resultado do operador subtração na expressão $4 - 2$ é 2 .

Propriedades dos Operadores

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

- Aridade
- Resultado
- Precedência
- Associatividade

A precedência indica a prioridade de aplicação de um operador em relação aos demais operadores que possam aparecer na expressão. Por exemplo, na expressão $2 + 3 \times 5$, o operador da multiplicação tem precedência maior, portanto é aplicado antes do que o operador da soma.

Propriedades dos Operadores

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

- Aridade
- Resultado
- Precedência
- Associatividade

A precedência indica a prioridade de aplicação de um operador em relação aos demais operadores que possam aparecer na expressão. Por exemplo, na expressão $2 + 3 \times 5$, o operador da multiplicação tem precedência maior, portanto é aplicado antes do que o operador da soma.

Nessa expressão, o primeiro operador aplicado é o de maior precedência, o da multiplicação. e seu resultado é 15. Então, para o computador, a expressão restante meio que se torna $2 + 15$.

Propriedades dos Operadores

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

- Aridade
- Resultado
- Precedência
- Associatividade

A precedência indica a prioridade de aplicação de um operador em relação aos demais operadores que possam aparecer na expressão. Por exemplo, na expressão $2 + 3 \times 5$, o operador da multiplicação tem precedência maior, portanto é aplicado antes do que o operador da soma.

Nessa expressão, o primeiro operador aplicado é o de maior precedência, o da multiplicação. e seu resultado é 15. Então, para o computador, a expressão restante meio que se torna $2 + 15$.

Só restou então a aplicação do operador da soma. Sabemos que seu resultado será 17, que é também o resultado da expressão inicial completa.

Propriedades dos Operadores

- Aridade
- Resultado
- Precedência
- Associatividade

Associatividade é a propriedade que define o desempate no caso da ocorrência de operadores com a mesma precedência em uma mesma expressão. A associatividade pode ser à direita ou à esquerda.

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

Propriedades dos Operadores

Para construir corretamente as expressões em C é importante conhecer os operadores e suas propriedades.

- Aridade
- Resultado
- Precedência
- Associatividade

O operador de multiplicação tem associatividade à esquerda. Ou seja, primeiro é aplicado o operador mais à esquerda da expressão e assim sucessivamente. Portanto, na expressão $2 \times 3 \times 5$, o primeiro operador aplicado é a multiplicação mais à esquerda e só depois a outra será aplicada com o resultado da aplicação do primeiro operador.

Associatividade é a propriedade que define o desempate no caso da ocorrência de operadores com a mesma precedência em uma mesma expressão. A associatividade pode ser à direita ou à esquerda.

Propriedades dos Operadores

- Aridade
- Resultado
- Precedência
- Associatividade

```
umInteiro = 8/2/2; //umInteiro receberá que valor?
```

Propriedades dos Operadores

- Aridade
- Resultado
- Precedência
- Associatividade

A resposta é **2**, mas precisamos conhecer todos os operadores aritméticos e suas propriedades para entender a resposta.

```
umInteiro = 8/2/2; //umInteiro receberá que valor?
```


Operadores e Expressões Aritméticos

Esses são os operadores aritméticos da linguagem C.

OPERADOR	SIGNIFICADO
-	menos unário (i.e., inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

Operadores e Expressões Aritméticos

Esses são os operadores aritméticos da linguagem C.

OPERADOR	SIGNIFICADO
-	menos [unário] (i.e., inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

Unário aqui se refere à aridade do operador, que no caso só precisa de um único operando para funcionar, justamente o operando que terá o sinal invertido.

Operadores e Expressões Aritméticos

Esses são os operadores aritméticos da linguagem C.

OPERADOR	SIGNIFICADO
-	menos unário (i.e., inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

O resultado da aplicação desse operador é o resto da divisão inteira do primeiro operando pelo segundo.

Operadores e Expressões Aritméticos

Esses são os operadores aritméticos da linguagem C.

OPERADOR	SIGNIFICADO
-	menos unário (i.e., inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

O resultado da aplicação desse operador é o resto da divisão inteira do primeiro operando pelo segundo.

Por exemplo, na expressão `5 % 2` o resultado do operador é `1`, pois é o que sobra efetuar a divisão inteira de 5 por 2.

Operadores e Expressões Aritméticos

Esses são os operadores aritméticos da linguagem C.

OPERADOR	SIGNIFICADO
-	menos unário (i.e., inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

O resultado da aplicação desse operador é o resto da divisão inteira do primeiro operando pelo segundo.

Por exemplo, na expressão `5 % 2` o resultado do operador é `1` pois é o que sobra efetuar a divisão inteira de 5 por 2.

Operadores e Expressões Aritméticos

Esses são os operadores aritméticos da linguagem C.

OPERADOR	SIGNIFICADO
-	menos unário (i.e., inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

O resultado da aplicação desse operador é o resto da divisão inteira do primeiro operando pelo segundo.

Por exemplo, na expressão `5%2` o resultado do operador é `1` pois é o que sobra efetuar a divisão inteira de 5 por 2.

Operadores e Expressões Aritméticos

Analise esses exemplos de utilização dos operadores e seus resultados.

Expressão	Resultado
$2.5 + 4$	6.5
$2.5 + 4.0$	6.5
$2 + 4$	6
$5 \% 2$	1
$5 / 2$	2
$5.0 / 2$	2.5

Operadores e Expressões Aritméticos

Analise esses exemplos de utilização dos operadores e seus resultados.

Expressão	Resultado
$2.5 + 4$	6.5
$2.5 + 4.0$	6.5
$2 + 4$	6
$5 \% 2$	1
$5 / 2$	2
$5.0 / 2$	2.5

Quando os dois operandos de um operador aritmético são números inteiros, o resultado também é um número inteiro.

Operadores e Expressões Aritméticos

Analise esses exemplos de utilização dos operadores e seus resultados.

Se *pelo menos* um dos operandos for um número real, o resultado também será um número real.

Expressão	Resultado
2.5 + 4	6.5
2.5 + 4.0	6.5
2 + 4	6
5 % 2	1
5 / 2	2
5.0 / 2	2.5

Quando os dois operandos de um operador aritmético são números inteiros, o resultado também é um número inteiro.

Operadores e Expressões Aritméticos

Analise esses exemplos de utilização dos operadores e seus resultados.

Se *pelo menos* um dos operandos for um número real, o resultado também será um número real.

Expressão	Resultado
$2.5 + 4$	6.5
$2.5 + 4.0$	6.5
$2 + 4$	6
$5 \% 2$	1
$5 / 2$	2
$5.0 / 2$	2.5

Quando os dois operandos de um operador aritmético são números inteiros, o resultado também é um número inteiro.

É isso que causa esses dois possíveis resultados da divisão de 5 por 2. Atenção para não errar na prova!

Operadores e Expressões Aritméticos

Essa é a tabela de precedência e associatividade dos operadores aritméticos.

OPERADOR	PRECEDÊNCIA	ASSOCIATIVIDADE
– (unário)	Alta (aplicado primeiro)	À direita
*, /, %	↓	À esquerda
+, – (binários)	Baixa (aplicado por último)	À esquerda

*use parênteses, como na matemática, para alterar a precedência e associatividade

Operadores e Expressões Aritméticos

Responda.

Expressão	Valor
$2 + 3 * 4$?
$(2 + 3) * 4$?
$2 * 3 / 6$?
$2 * (3 / 6)$?
$2 * (3.0 / 6)$?
$2 + 4 - 5 + -1$?
$6 / -2 + 4 * 2 - 2 + 1$?
$6 / (-2 + 4) * (-2 + 1)$?
$6 * 2 / 3 * 2$?
$(6 * 2) / (3 * 2)$?

Operadores e Expressões Aritméticos

Expressão	Valor
$2 + 3 * 4$	14
$(2 + 3) * 4$	20
$2 * 3 / 6$	1
$2 * (3 / 6)$	0
$2 * (3.0 / 6)$	1.0
$2 + 4 - 5 + -1$	0
$6 / -2 + 4 * 2 - 2 + 1$	4
$6 / (-2 + 4) * (-2 + 1)$	-3
$6 * 2 / 3 * 2$	8
$(6 * 2) / (3 * 2)$	2

Confira.

Entrada e Saída

- Usualmente, um programa precisa obter seus dados de entrada por meio de algum meio externo (por exemplo, teclado ou disco)
- De modo análogo, temos uma instrução que realiza a transferência dos dados de saída do programa para o meio de saída (por exemplo, tela ou impressora)

Entrada e Saída

- Usualmente, um programa precisa obter seus dados de entrada por meio de algum meio externo (por exemplo, teclado ou disco)
- De modo análogo, temos uma instrução que realiza a transferência dos dados com os resultados do programa para o meio de saída (por exemplo, tela ou impressora)

Veja o [vídeo com as instruções de saída](#).

Entrada e Saída

- Usualmente, um programa precisa obter seus dados de entrada por meio de algum meio externo (por exemplo, teclado ou disco)

Veja o [vídeo com as instruções de entrada](#).

- De modo análogo, temos uma instrução que realiza a transferência dos dados com os resultados do programa para o meio de saída (por exemplo, tela ou impressora)

Veja o [vídeo com as instruções de saída](#).

Entrada e Saída – Exemplos

```
scanf("%d", &a); //usuario digitou 5  
a = a + 5;  
printf("%d", a); //imprime o que?
```

```
a = a + 5;  
scanf("%d", &a); //usuario digitou 5  
a = 5 - a;  
printf("%d", a); //imprime o que?
```

```
scanf("%d", &a); //usuario digitou 5  
a = a - 4;  
a = 10 + 5;  
printf("%d", a); //imprime o que?
```

```
a = a + 5;  
scanf("%d", &a); //usuario digitou 5  
printf("%d", a); //imprime o que?
```

Entrada e Saída – Exemplos

```
scanf("%d", &a); //usuario digitou 5  
a = a + 5;  
printf("%d", a); //imprime o que?
```

Imprime **10**.

```
a = a + 5;  
scanf("%d", &a); //usuario digitou 5  
a = 5 - a;  
printf("%d", a); //imprime o que?
```

Imprime **0**.

```
scanf("%d", &a); //usuario digitou 5  
a = a - 4;  
a = 10 + 5;  
printf("%d", a); //imprime o que?
```

Imprime **15**.

```
a = a + 5;  
scanf("%d", &a); //usuario digitou 5  
printf("%d", a); //imprime o que?
```

Imprime **5**.

Agora você já consegue fazer um programa que leia três notas de um aluno e exiba a média.

Também consegue fazer o programa das latas de tinta.

Faça as atividades do Sala de Aula!