



# **Introdução à construção dos programas**

---

#include <biblioteca1.h>
#include <biblioteca2.h>
...
#include <biblioteca.h>

Outras definições
Outras definições

**int main (void) {**

instrução 1
instrução 2

instrução 3
instrução 4
. . .
instrução n

**}**

**Estrutura básica de todo código-fonte escrito em linguagem C**

```
#include <biblioteca1.h>
#include <biblioteca2.h>
...
#include <biblioteca.h>
```

Bibliotecas usadas pelo programa

```
Outras definições
Outras definições
```

Definições necessárias ao programa como constantes, variáveis globais, definição de tipos, funções e etc. Estas definições são opcionais

```
int main (void) {
    instrução 1
    instrução 2

    instrução 3
    instrução 4
    . . .
    instrução n
}
```

Na função main está a definição de todo o funcionamento do programa. As instruções podem ser definições de variáveis, expressões, chamada de funções, estruturas de controle e a instrução *return*

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

Inclusão das bibliotecas utilizadas

```
int main() {
```

```
float nota1, nota2, media;
```

```
int peso1 = 2, peso2 = 3;
```

Definição das variáveis

```
printf("Calculo da Media final \n");
```

Chamada de funções

```
nota1 = 7.5;
```

```
nota2 = 9.0;
```

```
media = (nota1*peso1 + nota2*peso2) / (peso1 + peso2);
```

Expressões

```
printf("Resultado = %.2f", media);
```

```
return 0;
```

```
}
```

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

```
int main() {  
    float nota1, nota2, media;  
    int peso1 = 2, peso2 = 3;
```

```
    printf("Calculo da Media final \n");
```

```
    nota1 = 7.5;
```

```
    nota2 = 9.0;
```

```
    media = (nota1*peso1 + nota2*peso2) / (peso1 + peso2);
```

```
    printf("Resultado = %.2f", media);
```

```
    return 0;
```

```
}
```

### Fluxo de Execução Natural do Programa

1. As instruções são executadas uma a uma
2. Na ordem em que aparecem
3. Uma única vez cada uma

**Para iluminar de maneira correta os cômodos de uma casa, para cada  $m^2$ , deve-se usar 18W de potência. Qual é o algoritmo para determinar a potência de iluminação que deverá ser utilizada para iluminar um cômodo?**

**Algoritmo:**

1. Pedir as dimensões do cômodo
2. Calcular a área do cômodo
3. Calcular a potência de iluminação
4. Mostrar o resultado

**Quais seriam os tipos de instruções na linguagem de programação necessários para a implementação do algoritmo?**

**Instruções do programa:**

1. Instruções para ler valores do teclado – *vamos usar para ler as dimensões do cômodo.*
2. Instruções para armazenar valores na memória – *tanto para armazenar os valores lidos quanto para armazenar o resultado dos cálculos.*
3. Instruções para efetuar cálculos – *precisamos construir expressões matemáticas com os valores lidos.*
4. Instruções para informar o resultado – *para exibir na tela o resultado do programa.*

# Variáveis

- Um programa recebe dados que precisam ser armazenados no computador para serem utilizados no processamento. Esse armazenamento é feito na **memória**
- Uma variável possui nome e tipo, e representa uma posição de memória
- O conteúdo da variável é modificado ao longo do tempo durante a execução do programa
- Embora uma variável possa assumir diferentes valores, ela só armazena um valor a cada instante

# Variáveis

Podemos imaginar a memória do computador como um armário. Quando o armário está vazio, suas gavetas estão livres e não armazenam nenhum dado



Uma variável representa uma gaveta do armário

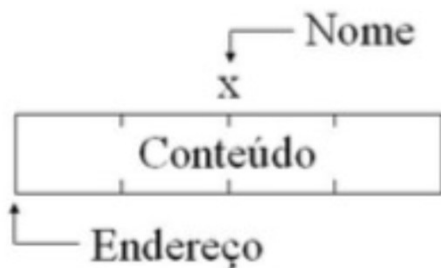
Através das variáveis nós conseguimos armazenar e manipular valores na memória do computador. Cada variável ocupa um espaço exclusivo na memória

Toda variável armazena um **valor**. Dependendo do tipo de valor armazenado, as variáveis podem ocupar mais ou menos espaço da memória



# Variáveis

- Cada variável deve possuir um **nome**
- O nome de uma variável funciona como um rótulo para o valor 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



Cada variável também possui um **endereço**, que é a posição na memória onde ela 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

```
int x;  
float y;
```

Definição de uma variável:

Começamos pela especificação do tipo de dado que a variável irá armazenar seguido do seu nome, separados com um espaço.

Por enquanto vamos usar os tipos:

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

Memória



# Variáveis e atribuições

```
int x;  
float y;
```

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



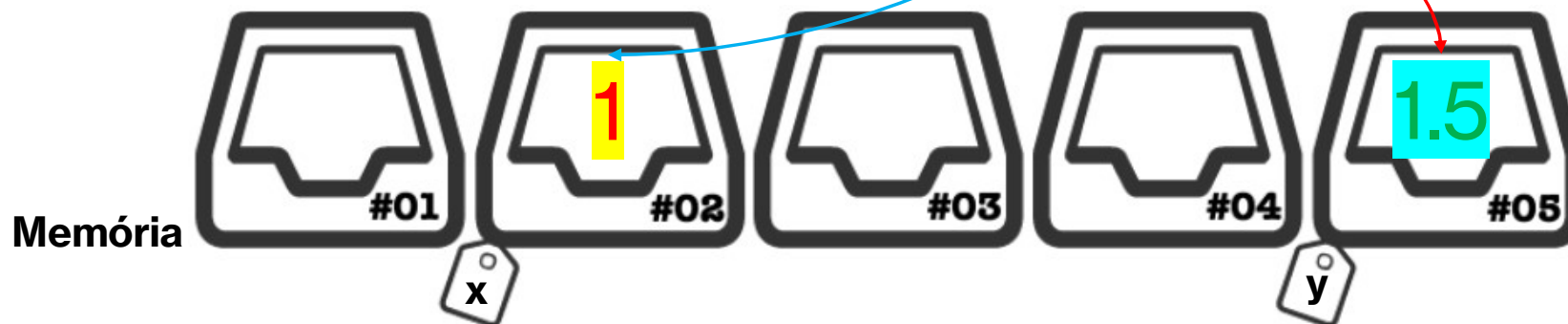
# Variáveis e atribuições

```
int x;  
float y;
```

```
x = 1;  
y = 1.5;
```

A instrução de atribuição, representada pelo =, define o valor da variável, ou seja, o conteúdo armazenado no espaço dela na memória

Devemos atribuir valores que correspondam ao tipo definido na variável



# Variáveis e atribuições

- Define-se a variável antes de usá-la. É uma boa prática iniciar o código com a definição de todas as variáveis.

```
int numero;  
float desconto, preco;  
char umaLetra;
```

Dê nomes significativos para as variáveis, nomes que indiquem o que elas representam

```
numero      = 5;  
desconto    = 1.20 + 0.30;  
preco       = 5.50 - desconto;  
umaLetra    = 'A';
```

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

```
int numero;  
float desconto, preco;  
char umaLetra;
```

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

```
numero = 5;  
desconto = 1.20 + 0.30;  
preco = 5.50 - desconto;  
umaLetra = 'A';
```

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

O lado direito da atribuição 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 numero;  
float desconto, preco;  
char umaLetra;
```

```
numero      = 5;  
desconto    = 1.20 + 0.30;  
preco       = 5.50 - desconto;  
umaLetra    = 'A';
```

Quando o nome de uma variável aparecer em uma expressão, será sempre considerado o valor que ela armazena no momento da execução da instrução.

Toda instrução em C termina com um ;

# Atribuições e Expressões

```
int numero;  
char umaLetra;  
float nota1, nota2, media;
```

```
numero = 1;  
umaLetra = 'A';  
nota1 = 8.5;  
nota2 = nota1 + 1;  
media = (nota1 + nota2) / 2;  
numero = numero + 2;
```

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

```
numero: ?  
umaLetra: ?  
nota1: ?  
nota2: ?  
media: ?
```

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



# Operadores e expressões

- Expressões aritméticas
- Expressões relacionais
- Expressões lógicas

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

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

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

# Propriedade dos operadores

- Para construir expressões corretas em C é importante conhecer os operadores e suas propriedades:

- Aridade
- Resultado
- Precedência
- Associatividade

Aridade indica o número de operandos necessários para a utilização do operador. Por exemplo, a soma tem aridade 2 pois precisa de dois operandos, como em **5 + 7**.

Resultado é o valor gerado pela aplicação do operador. Por exemplo, o resultado do operador subtração na expressão **4 - 1** é **3**.

# Propriedade dos operadores

- Para construir expressões corretas 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  $2 + 3 * 5$ , 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 é equivalente a uma expressão  $2 + 15$ . A seguir acontece a aplicação do operador da soma, cujo resultado será **17**, que é também o resultado da expressão inicial completa.

# Propriedade dos operadores

- Para construir expressões corretas em C é importante conhecer os operadores e suas propriedades:

- 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.

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  $2 * 3 * 5$ , 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.

# Operadores e expressões aritméticas

Operador	Significado
-	menos <u>unário</u> (inversão de sinal)
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

**Unário** 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.

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

Exemplo, na expressão **5 % 2** o resultado do operador é **1**, pois é o que sobra ao efetuar a divisão inteira de 5 por 2.

# Operadores e expressões aritméticas

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 aritméticas

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

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

# Operadores e expressões aritméticas

- Tabela de precedência e associatividade dos operadores aritméticos.

Operador	Precedência	Associatividade
- (unário)	<b>Alta</b> (aplicado primeiro)	À direita
* / %	↓	À esquerda
+ - (binários)	<b>Baixa</b> (aplicado por último)	À esquerda

Use parênteses, como na matemática, para alterar a precedência e associatividade



# Operadores e expressões aritméticas

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 aritméticas

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

# Entrada e saída

- Entrada de dados – são instruções que permitem que o programa receba dados, de algum meio externo (por exemplo, o teclado), para serem processados
- Saída de dados – são instruções que permitem que o programa mostre os resultados, através de um meio de saída (por exemplo, a tela), que foram calculados

# Instruções de Saída

- **puts**

- Exibe um texto na tela



The diagram shows the code `puts("Meu primeiro programa em C");` inside a blue rectangular box. The word `puts` is enclosed in a dashed blue box, and the string `"Meu primeiro programa em C"` is enclosed in a dashed red box. A blue arrow points from the text 'Exibe um texto na tela' to the `puts` function. A red arrow points from the text 'Para dizer ao puts qual mensagem será exibida, é preciso passar esse valor nos parâmetros. Os parâmetros passados às funções ficam entre parênteses' to the opening parenthesis of the string. A green arrow points from the text 'Cada função tem uma quantidade e tipos de parâmetros específicos. A função puts só exige um parâmetro do tipo texto e é por isso que ele está entre aspas' to the closing parenthesis of the string.

```
puts("Meu primeiro programa em C");
```

O **puts** é uma *função* disponível da linguagem C. Para executar uma função nós utilizamos o nome dela

Para dizer ao **puts** qual mensagem será exibida, é preciso passar esse valor nos parâmetros. Os parâmetros passados às funções ficam entre parênteses

Cada função tem uma quantidade e tipos de parâmetros específicos. A função **puts** só exige um parâmetro do tipo texto e é por isso que ele está entre aspas

# Instruções de Saída

Este é o código-fonte completo de um programa que exibe três mensagens. Perceba que as instruções estão sempre *dentro* da **main**

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

```
int main (){
```

```
    puts ("Meu primeiro programa em C.");  
    puts ("Este eh um exemplo com a função puts.");  
    puts ("Fim do programa.");
```

```
    return 0;
```

```
}
```

O **puts** e as demais instruções de entrada e saída estão definidos na biblioteca **stdio.h**, sendo necessária a sua inclusão para utilização.

Um detalhe importante do funcionamento do **puts** é que ele pula para a próxima linha após exibir a mensagem.

# Instruções de Saída

Assim, ao compilar e executar o código-fonte do exemplo, o programa apresentará a seguinte saída:

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

```
int main (){
```

```
    puts ("Meu primeiro programa em C.");  
    puts ("Este eh um exemplo com a função puts.");  
    puts ("Fim do programa.");
```

```
    return 0;
```

```
}
```

```
Meu primeiro programa em C.  
Este eh um exemplo com a função puts.  
Fim do programa.
```

# Instruções de Saída

- **printf**
  - Outra instrução de saída
  - Exibe texto e conteúdo de variável na tela

```
printf ("Meu primeiro programa em C.");  
printf ("Este eh um exemplo com a função printf.");  
printf ("Fim do programa.");
```

Substituí as chamadas do puts pelo printf.

O **printf** serve para o mesmo propósito de exibir uma mensagem, e é utilizado da mesma forma que o **puts**.

# Instruções de Saída

Detalhe importante: o ***printf*** não pula linha após exibir a mensagem. Sendo assim, a saída do programa do exemplo é a seguinte:

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

```
int main (){
```

```
Meu primeiro programa em C. Este eh um exemplo com a função printf. Fim do programa.
```

```
printf ("Meu primeiro programa em C.");
```

```
printf ("Este eh um exemplo com a função printf.");
```

```
printf ("Fim do programa.");
```

```
return 0;
```

```
}
```



# Instruções de Saída

Para que o ***printf*** pule linhas basta adicionar um **`\n`** no texto do parâmetro no local onde se deseja a quebra de linha.

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

```
int main (){
```

```
    printf ("Meu primeiro programa em C.\n");
```

```
    printf ("Este eh um exemplo com a função printf.\n");
```

```
    printf ("Fim do programa. \n");
```

```
    return 0;
```

```
}
```

Apesar de ser composta por dois caracteres, a *sequência de escape* **`\n`** é considerada um único caractere especial e indica uma quebra de linha no local em que ele aparece.

```
Meu primeiro programa em C.
Este eh um exemplo com a função printf.
Fim do programa.
```

# Instruções de Saída

Com o ***printf*** também é possível mostrar o conteúdo de variáveis

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

```
int main(){
```

```
    int numero;
```

```
    numero = 23;
```

```
    printf ("O numero eh = %d\n", numero);
```

```
    return 0;
```

```
}
```

Esse código-fonte define a variável *numero* e atribui o valor 23 a ela.

# Instruções de Saída

Com o ***printf*** também é possível mostrar o conteúdo de variáveis

```
#include <stdio.h>

int main(){
    int numero;

    numero = 23;
    printf ("O numero eh = %d\n", numero);
    return 0;
}
```

A instrução ***printf*** agora apresenta dois parâmetros. Os parâmetros devem ser separados por vírgulas e o primeiro deles é sempre um texto entre aspas.

Atenção à essa sequência especial, ela é um **especificador de formato**. Todo especificador de formato começa com o sinal de porcentagem.

# Instruções de Saída

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

```
int main(){
```

```
    int numero;
```

```
    numero = 23;
```

```
    printf ("O numero eh = %d\n", numero);
```

```
    return 0;
```

```
}
```

Existe um especificador de formato para cada tipo:

**%d** é para números inteiros

**%c** é para caracteres e

**%f** para **float** (números reais).

A ocorrência do especificador exigiu a presença do segundo parâmetro do **printf**. Esse parâmetro vai definir justamente qual é o valor que deve ser exibido ao substituir o especificador no texto.

```
O numero eh = 23.
```

# Instruções de Saída

É permitido inserir quantos especificadores de formato forem necessários. Para cada especificador deve-se existir um parâmetro adicional com os valores que substituirão os especificadores no texto.

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

```
int main(){
```

```
    int numero;
```

```
    numero = 23;
```

```
    printf ("O numero eh = %d e o seu antecessor eh = %d.\n", numero, numero - 1);
```

```
    return 0;
```

```
}
```

Os valores dos parâmetros devem respeitar a ordem e os tipos especificados e podem ser variáveis, valores constantes ou expressões.

```
O numero eh = 23 e o seu antecessor eh = 22.
```

# Instruções de Entrada

- scanf
  - é a principal instrução para entrada de dados da linguagem C

O primeiro parâmetro será sempre um texto. O texto deve conter apenas um **especificador de formato**

```
scanf ("%d", &numero);
```

O segundo parâmetro deve ser o nome da **variável** que armazenará o valor a ser lido do teclado. No scanf o nome da variável deve ser sempre precedido do **&**

# Instruções de Entrada

```
printf ("Digite um número: ");  
scanf ("%d", &numero);
```

Quando o programa é executado, ele permanece parado na instrução ***scanf*** até que o usuário digite um valor e pressione a tecla *ENTER*

É importante sempre adicionar um *prompt* antes do ***scanf***, ou seja, uma mensagem informando o que deve ser digitado

O ***scanf*** não precisa do *prompt* para funcionar, mas ele é necessário para informar ao usuário que o programa espera que ele digite um valor de entrada para continuar. Sem esta mensagem o usuário não saberia o que fazer e o programa ficaria parado

Observe que como o ***scanf*** especificou a leitura de um inteiro (com o ***%d***), foi passado como parâmetro uma variável do tipo inteiro

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

```
int main(){
```

```
    int numero;
```

```
    printf ("Digite um numero: ");
```

```
    scanf ("%d", &numero);
```

```
    printf ("O numero eh = %d e o seu antecessor eh = %d.\n", numero, numero - 1);
```

```
    return 0;
```

```
}
```

Ao compilar e executar, o programa exibirá o valor inteiro digitado no teclado juntamente com o restante da mensagem especificada no ***printf***.

```
Digite um numero: 16
O numero eh = 16 e o seu antecessor eh = 15.
```



# Entrada e Saída: Exemplos

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

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

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

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

# Entrada e Saída: Exemplos

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

Mostra **10**

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

Mostra **0**

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

Mostra **15**

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

Mostra **5**

**Para iluminar de maneira correta os cômodos de uma casa, para cada  $m^2$ , deve-se usar 18W de potência. Qual é o algoritmo para determinar a potência de iluminação que deverá ser utilizada para iluminar um cômodo?**

Instruções do programa:

1. Instruções para ler valores do teclado – *vamos usar para ler as dimensões do cômodo.*
2. Instruções para armazenar valores na memória – *tanto para armazenar os valores lidos quanto para armazenar o resultado dos cálculos.*
3. Instruções para efetuar cálculos – *precisamos construir expressões matemáticas com os valores lidos.*
4. Instruções para informar o resultado – *para exibir na tela o resultado do programa.*

```
/*
```

Para iluminar de maneira correta os cômodos de uma casa, para cada  $m^2$ , deve-se usar 18W de potência. Qual é o algoritmo para determinar a potência de iluminação que deverá ser utilizada para iluminar um cômodo?

```
*/
```

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

```
int main(){
```

```
    float largura, comprimento, area, potencia;
```

```
    printf ("Digite a largura do comodo: ");
```

```
    scanf ("%f", &largura);
```

```
    printf ("Digite o comprimento do comodo: ");
```

```
    scanf ("%f", &comprimento);
```

```
    area = largura * comprimento;
```

```
    potencia = 18 * area;
```

```
    printf ("Area do comodo = %f\nPotencia de iluminacao necessaria = %f.\n", area, potencia);
```

```
    return 0;
```

```
}
```

```
/*
```

Para iluminar de maneira correta os cômodos de uma casa, para cada  $m^2$ , deve-se usar 18W de potência. Qual é o algoritmo para determinar a potência de iluminação que deverá ser utilizada para iluminar um cômodo?

```
*/
```

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

```
int main(){
```

```
    float largura, comprimento, area, potencia;
```

```
    printf ("Digite a largura do comodo: ");
```

```
    scanf ("%f", &largura);
```

```
    printf ("Digite o comprimento do comodo: ");
```

```
    scanf ("%f", &comprimento);
```

```
    area = largura * comprimento;
```

```
    potencia = 18 * area;
```

```
    printf ("Area do comodo = %f\nPotencia de iluminacao necessaria = %f.\n", area, potencia);
```


```
    return 0;
```

```
}
```

```
Digite a largura do comodo: 1.5
Digite o comprimento do comodo: 2
Area do comodo = 3.00m2
Potencia de iluminacao necessaria = 54.00 W.
```

O programa executa uma instrução por vez, uma única vez e na ordem em que aparecem.

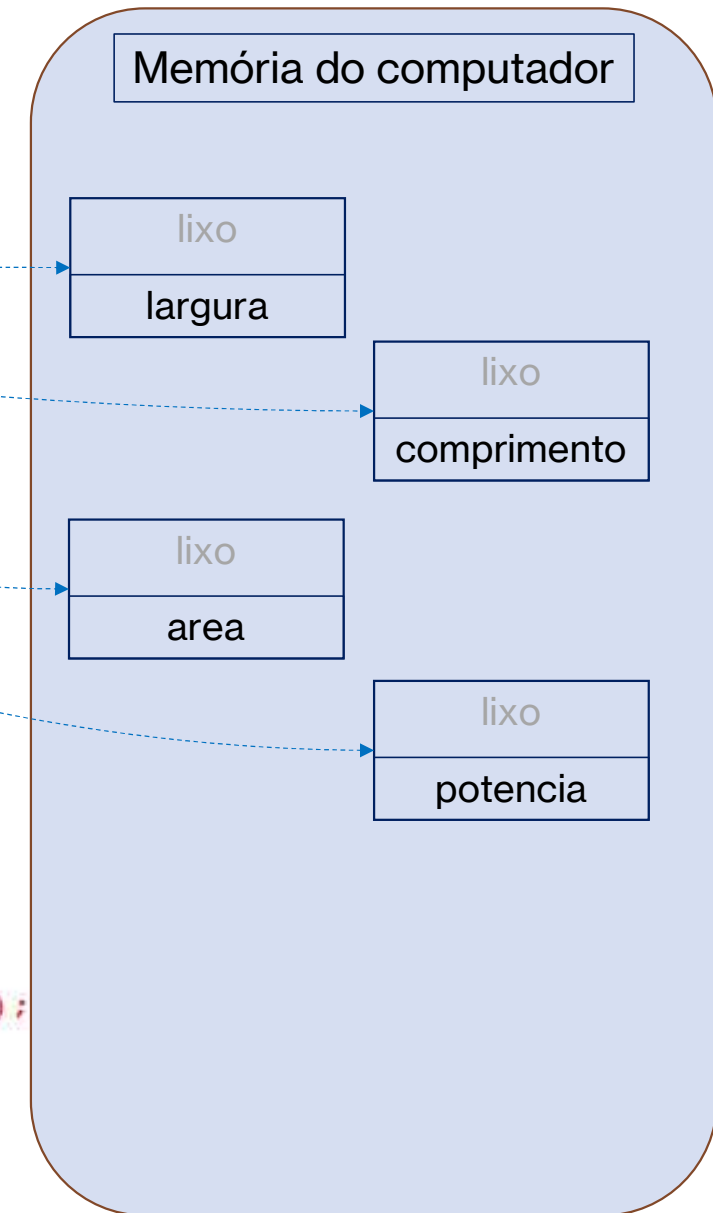
**Fluxo de execução  
do programa**



```
1  #include <stdio.h>
2
3  int main() {
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13            "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

O programa inicia a execução pela função main() e na linha 4 há a definição de quatro variáveis que são alocadas na memória

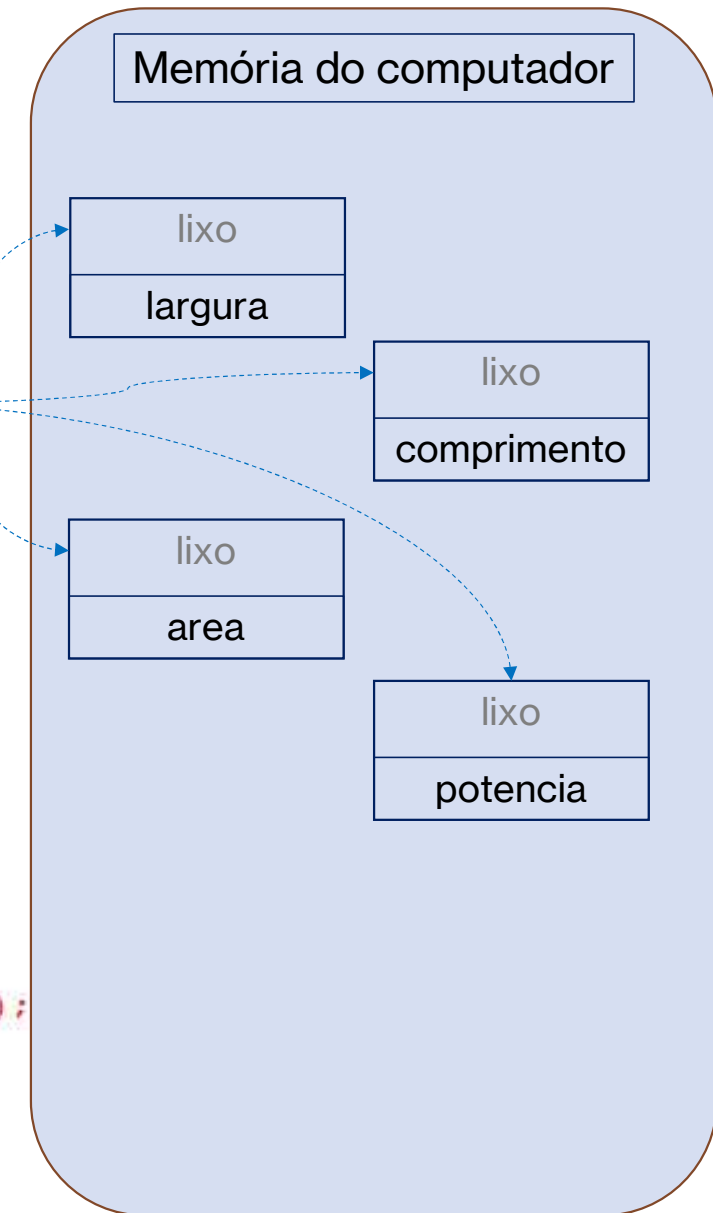
```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13            "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```



O programa inicia a execução pela função `main()` e na linha 4 há a definição de quatro variáveis que são alocadas na memória

Quando são alocadas na memória, as variáveis armazenam um valor qualquer que não pode ser previsto, chamamos esse valor de *lixo de memória*.

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13            "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

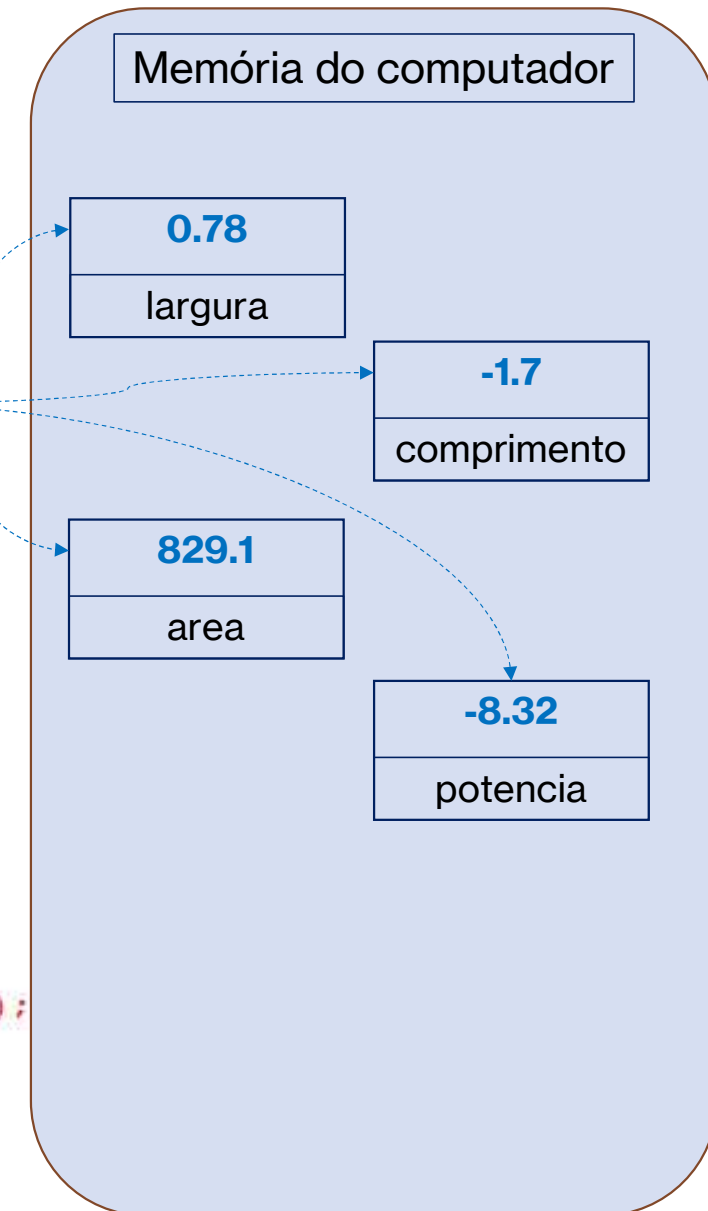




O programa inicia a execução pela função `main()` e na linha 4 há a definição de quatro variáveis que são alocadas na memória

Quando são alocadas na memória, as variáveis armazenam um valor qualquer que não pode ser previsto, chamamos esse valor de *lixo de memória*.

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```



Na linha 6 o programa mostrará na tela a mensagem especificada no **printf**

Na linha 7 será lido um valor **float** e o valor digitado será armazenado na variável **largura**. Vamos supor que seja digitado o valor 1.5

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

0.78

largura

-1.7

comprimento

829.1

area

-8.32

potencia

Na linha 6 o programa mostrará na tela a mensagem especificada no **printf**

Na linha 7 será lido um valor **float** e o valor digitado será armazenado na variável **largura**. Vamos supor que seja digitado o valor 1.5

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

1.5

largura

-1.7

comprimento

829.1

area

-8.32

potencia

Na linha 8 o programa mostrará na tela outra mensagem especificada no **printf**

Na linha 9 será lido um segundo valor **float** e esse valor será armazenado na variável **comprimento**. Vamos supor que seja digitado o valor 2

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

1.5

largura

-1.7

comprimento

829.1

area

-8.32

potencia

Na linha 8 o programa mostrará na tela outra mensagem especificada no **printf**

Na linha 9 será lido um segundo valor **float** e esse valor será armazenado na variável **comprimento**. Vamos supor que seja digitado o valor 2

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

1.5

largura

2

comprimento

829.1

area

-8.32

potencia



A linha 10 executa o comando de atribuição. O resultado calculado ao lado direito do = será armazenado na variável **area**

A expressão multiplica os valores armazenados nas variáveis **largura** e **comprimento**

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13            "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

1.5

largura

2

comprimento

829.1

area

-8.32

potencia

A linha 10 executa o comando de atribuição. O resultado calculado ao lado direito do = será armazenado na variável **area**

A expressão multiplica os valores armazenados nas variáveis **largura** e **comprimento**

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

O resultado **3.0** é armazenado na variável **area**

### Memória do computador

1.5

largura

2

comprimento

3.0

area

-8.32

potencia

De forma similar, a atribuição da linha 11 é calculada multiplicando 18 pelo conteúdo da variável **area**

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

1.5

largura

2

comprimento

3.0

area

-8.32

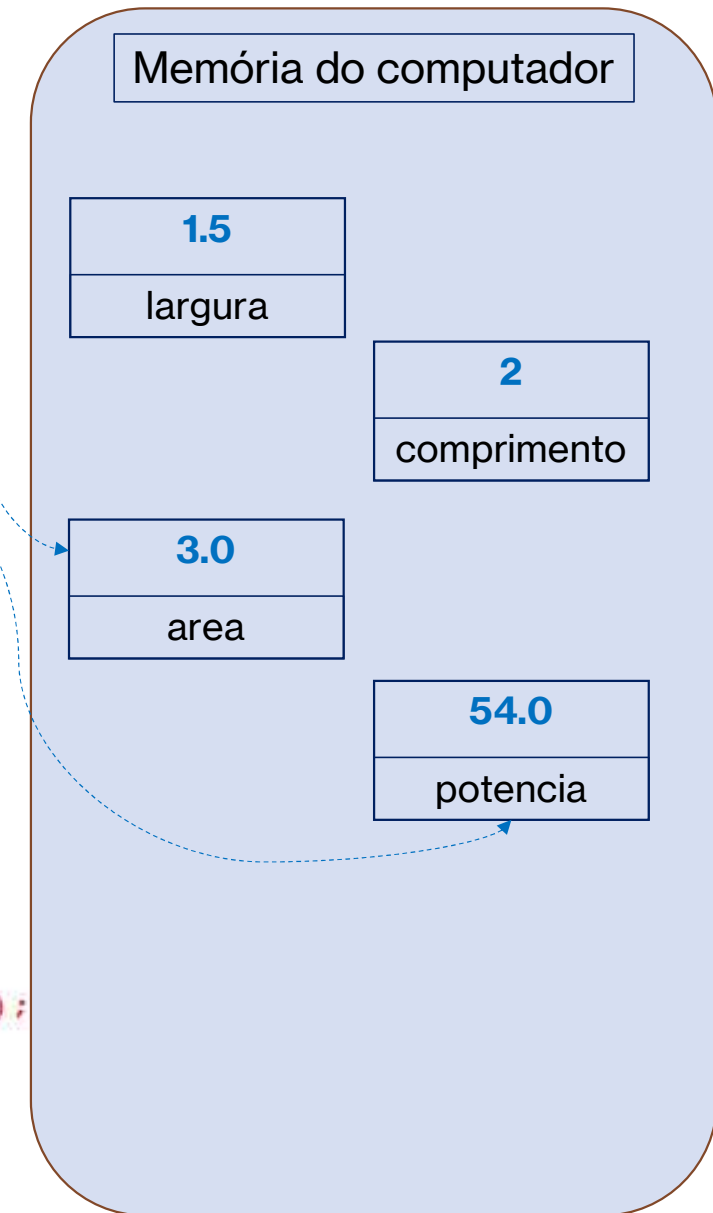
potencia



De forma similar, a atribuição da linha 11 é calculada multiplicando 18 pelo conteúdo da variável **area**

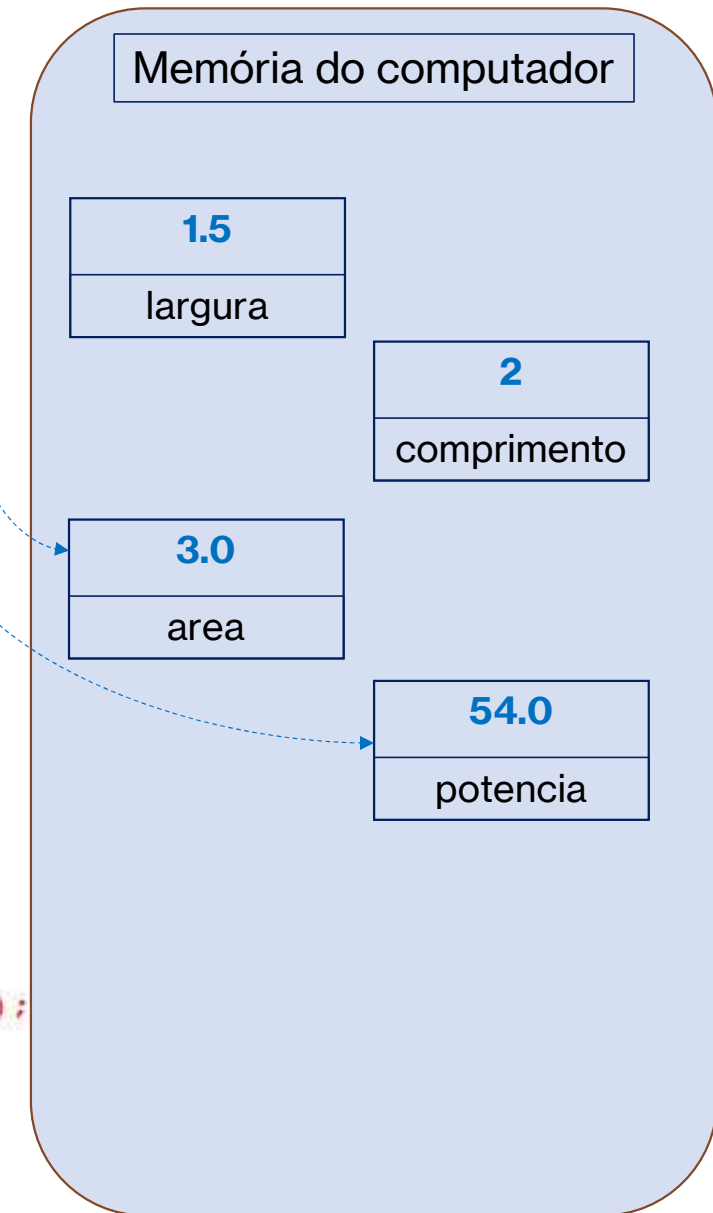
O resultado 54.0 é armazenado na variável **potencia**

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```



A execução do **printf** na linha 12 mostrará na tela o valor armazenado na variável **area** e o valor da variável **potencia**, que é a saída do programa.

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```



O programa termina com a execução do comando return na linha 15.

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

### Memória do computador

1.5

largura

2

comprimento

3.0

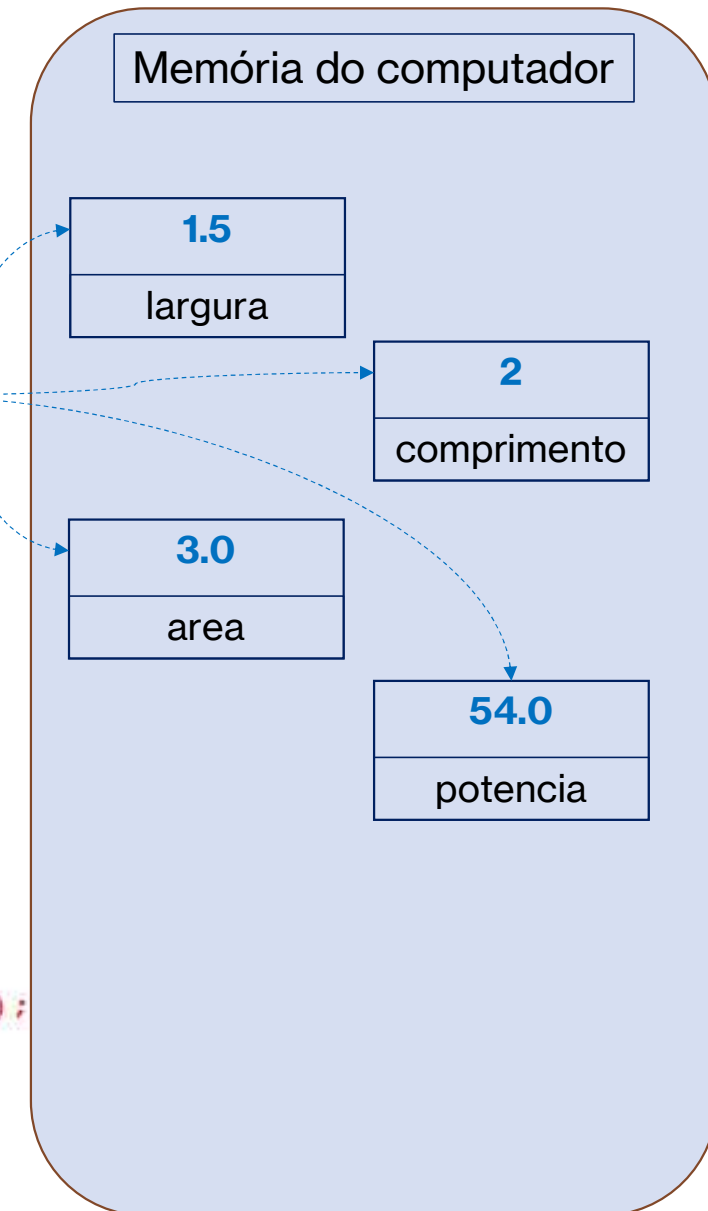
area

54.0

potencia

Depois que o programa é finalizado, todas as variáveis são liberadas da memória

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13            "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```



O espaço livre poderá ser usado por outras variáveis de outros programas

Memória do computador

```
1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13            "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }
```

```

1  #include <stdio.h>
2
3  int main(){
4      float largura, comprimento, area, potencia;
5
6      printf ("\n\nDigite a largura do comodo: ");
7      scanf ("%f", &largura);
8      printf ("Digite o comprimento do comodo: ");
9      scanf ("%f", &comprimento);
10     area = largura * comprimento;
11     potencia = 18 * area;
12     printf ("Area do comodo = %.2fm2\nPotencia de "
13             "iluminacao necessaria = %.2f W.\n", area, potencia);
14
15     return 0;
16 }

```

Porém, o lixo fica

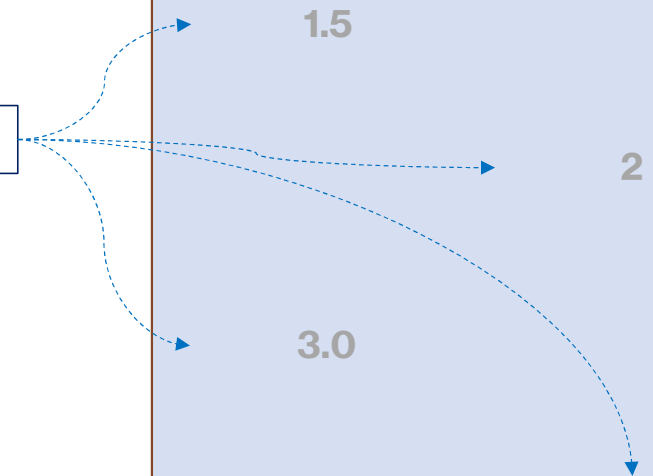
Memória do computador

1.5

2

3.0

54.0





# Qualidades de um bom programa

- A função do programador não é apenas resolver um problema com um programa, mas sim de fazê-lo da melhor maneira possível
- Além do planejamento inicial antes de começar a programação, é importante observar algumas características esperadas em um bom programa

• Legibilidade	• Reusabilidade
• Manutenibilidade	• Robustez
• Portabilidade	• Usabilidade
• Eficiência	• Confiabilidade

- Legibilidade - Um código-fonte de um bom programa deve ser fácil de ser lido e compreendido por outro programador. Um programa legível é bem organizado estruturalmente, bem indentado, suas variáveis possuem nomes significativos, as expressões longas contêm parênteses, etc.
- Manutenibilidade - Um código-fonte deve ser manutenível, ou seja, tanto o programador original quanto outros programadores devem ser capazes de realizar manutenções nesse código sem muitas dificuldades. As manutenções podem incluir correções de erros, correções de novas funcionalidades, etc. Além da legibilidade, a escolha de soluções, instruções e uma estrutura mais simples e clara tornam o código manutenível.
- Portabilidade - Um código-fonte portátil permite sua compilação em todas ou muitas das plataformas disponíveis sem precisar de modificações (Windows, Linux, macOS, iOS, Android, Arduino, ...). Seria uma utopia a compilação em todas as plataformas, mas é bom tentar pelo menos as principais ou mais importantes. Algumas instruções do C não são portáveis – ou seja, não funcionam em todas as plataformas – e devemos evitá-las. Normalmente a documentação da linguagem traz essa informação.



- Eficiência - Um programa é mais eficiente que outro quando ele resolve o mesmo problema em menos tempo (efetuando menos instruções) e utilizando menos memória (definindo menos variáveis). Devemos sim nos preocupar com a eficiência, mas não nesse momento. Por enquanto vamos nos preocupar primeiro em solucionar os problemas e quando estivermos mais experientes trabalhamos a eficiência.
- Reusabilidade - Um código-fonte reusável apresenta instruções e sequências de instruções relativamente genéricas, que podem ser facilmente copiadas para utilização em outros programas.
- Robustez – Um programa robusto é resiliente para continuar funcionando mesmo em situações não esperadas. Por exemplo quando alguém digita um nome enquanto o programa esperava uma idade. Outras situações adversas também podem ocorrer, como a remoção de um arquivo necessário enquanto o programa executava. O programador deve prever a maioria dessas situações e incluir no código instruções capazes de contorná-las.

- Usabilidade - Os programas devem apresentar uma interface simples e intuitiva para as pessoas. Um usuário deve sempre saber como interagir corretamente com o programa sem dúvidas.
- Confiabilidade - A confiabilidade é um pouco mais difícil de mensurar. Mas desejamos um programa em que possamos confiar, que tenhamos certeza que as respostas apresentadas são sempre corretas. Além disso, um programa confiável deve sempre apresentar uma resposta quando inserimos as informações de entrada, não apresentando erros ou fechamento abrupto em situações que deveria funcionar normalmente.

# Exercícios práticos

1. Faça um programa que receba três notas, calcule e mostre a média aritmética das notas.
2. Faça um programa que receba três notas e seus respectivos pesos, calcule e mostre a média ponderada das notas.
3. Faça um programa que receba um número positivo e maior que zero, calcule e mostre:
  - a. O número digitado ao quadrado;
  - b. O número digitado ao cubo;
  - c. A raiz quadrada do número digitado;
  - d. A raiz cúbica do número digitado.
4. Faça um programa que calcule e mostre a área de um círculo. Sabe-se que:  $\text{Área} = \pi R^2$ .
5. Faça um programa que receba dois números inteiros, calcule e mostre:
  - a. A divisão do primeiro número pelo segundo;
  - b. O resto da divisão entre eles.

# Exercícios práticos

6. Fazer um programa que calcule e mostre a área e o volume de um cilindro. Sabe-se que  $A = 2 \pi r (h+r)$  e  $V = \pi r^2 h$ .
7. Faça um programa que receba um número inteiro representando a quantidade de segundos. Calcule e mostre a quantidade de horas, minutos e segundos que esse número representa. Por exemplo, o número 5984 corresponde a 1 hora 39 minutos e 44 segundos.
8. Sabe-se que o quilowatt de energia custa  $1/5$  do salário mínimo. Faça um programa que receba o valor do salário mínimo e a quantidade de quilowatts consumida por uma residência. Calcule e mostra:
  - O valor de cada quilowatt;
  - O valor a ser pago por essa residência;
  - O valor a ser pago com desconto de 15%

# Exercícios práticos

9. Faça um programa que receba o número de horas trabalhadas, o valor do salário mínimo e o número de horas extras trabalhadas. Calcule e mostre o salário a receber, de acordo com as regras a seguir:
  - A hora trabalhada vale  $\frac{1}{8}$  do salário mínimo;
  - A hora extra vale  $\frac{1}{4}$  do salário mínimo;
  - O salário bruto equivale ao número de horas trabalhadas multiplicado pelo valor da hora trabalhada;
  - A quantia a receber pelas horas extras equivale ao número de horas extras trabalhadas multiplicado pelo valor da hora extra;
  - O salário a receber equivale ao salário bruto mais a quantia a receber pelas horas extras.
  
10. Faça um programa que leia o salário bruto de um funcionário, calcule e mostre o seu salário líquido. Sabe-se que:
  - O salário bruto teve um reajuste de 38%;
  - O funcionário receberá uma gratificação de 20% do salário bruto;
  - O salário total é descontado em 15%.