Algoritmos e Linguagem de Programação

- Capítulo 3
- Programação Sequencial

- Prof. Me. Renato Carioca Duarte

Programação Sequencial - Objetivos

- Apresentar detalhes relacionados a tipos de dados usados na linguagem C#.
- Descrever o conceito e a aplicação de variáveis.
- Mostrar os operadores aritméticos e as expressões aritméticas usando os operadores.
- Descrever a estrutura básica de um programa escrito na linguagem.
- Demonstrar o uso de programas sequenciais permeando as ações de entrada, processamento matemático e saída com cadeias de caracteres, valores inteiros e valores reais.

Tipos de Dados

- Os dados que usamos em um computador representam os elementos do mundo exterior, que são as informações manipuladas por nós seres humanos.
- Eles devem primeiramente serem abstraídos e então processados. Podem ser categorizados em três tipos primitivos ou básicos:
  - numéricos (representados por valores numéricos inteiros ou reais),
  - caracteres (valores alfabéticos ou alfanuméricos),
  - lógicos (valores binários, como por exemplo, falso e verdadeiro).
- A linguagem C# nos oferece um conjunto de tipos de dados predefinidos, classificados em: numérico inteiro, numérico de ponto flutuante, caractere e lógico.
- **Os tipos de dados são usados a fim de alocar espaço de memória para armazenamento e manipulação de valores de trabalho de um programa.**

Tipos de Dados

C# Type	Valores possíveis de se armazenar
bool	Verdadeiro ou Falso (Valores booleanos)
byte	0 a 255 (8 bits)
sbyte	-128 a 127 (8 bits)
char	Um caractere (16 bits)
decimal	±1.0 × 10−28 a ±7.9 × 1028 (128 bits)
double	±5.0 × 10−324 a ±1.7 × 10308 (64 bits)
float	±1.5 × 10−45 a ±3.4 × 1038 (32 bits)
int	-2,147,483,648 a 2,147,483,647 (32 bits)
uint	0 a 4,294,967,295 (32 bits)
long	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 (64 bits)
ulong	0 a 18,446,744,073,709,551,615 (64 bits)
object	Qualquer tipo
short	-32,768 a 32,767 (16 bits)
ushort	0 a 65,535 (16 bits)
string	Sequência de caracteres (16 bits por caractere)

Variáveis

- **Toda informação a ser processada em um computador por um programa necessita ser previamente armazenada na memória.**
- Conseguimos executar essa ação quando usamos variáveis. Uma variável é uma região de memória, previamente identificada por um rótulo (nome), que tem por finalidade armazenar os dados de um programa temporariamente.
- **Cada variável armazena apenas um valor por vez**, sendo esse valor um elemento qualificado a partir de um dos tipos de dados da linguagem.
- O nome de uma variável é usado para sua identificação e posterior manipulação dentro do programa.

Variáveis

- **O nome de uma variável é usado para sua identificação e posterior manipulação dentro do programa.**
- Para usar variáveis, precisamos considerar algumas regras quanto aos nomes que podemos definir. São elas:
  - Nome de variável pode ser atribuído com um ou mais caracteres.
  - O primeiro caractere do nome de uma variável não pode ser em nenhuma hipótese um número; sempre deve ser uma letra.
  - O nome de uma variável não pode ter espaços em branco.
  - Não pode ser nome de uma variável uma palavra reservada a uma instrução ou um identificador de uma linguagem de programação, bem como o nome de seus controles.
  - Não podem ser utilizados outros caracteres que não letras e números, com exceção do caractere underscore "_", que pode ser utilizado para simular a separação de duas palavras, como Nome_Aluno, que também podem estar escritas como NomeAluno.

Variáveis

- Dentro de um programa uma variável pode exercer dois papéis:
  - um de ação, quando ela é modificada ao longo de um programa para apresentar um determinado resultado.
  - outro de controle, quando ela é vigiada e controlada durante a execução de um programa (esse tipo de variável será estudado nos capítulos que abordam a tomada de decisões e o uso de laços).

- Todo dado a ser armazenado na memória de um computador por meio de uma variável deve ser analisado previamente, ou seja, **primeiramente é necessário saber o seu tipo para depois fazer o seu armazenamento**.

- Uma vez armazenado, o dado pode ser utilizado e manipulado a qualquer momento durante a execução do programa.

Exemplo de Declaração de Variáveis

```
int num;
double x, y;
char l;
bool resp;
string nome;
```

Operadores Aritméticos

- Os operadores aritméticos podem ser unários ou binários.
- São binários quando atuam em operações de multiplicação, divisão, adição e subtração, em que se utilizam dois componentes.
- São unários quando atuam na inversão de um valor, atribuindo a ele o sinal positivo ou negativo, ou seja, atuam diretamente em apenas um componente.

Operador Aritmético	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão)

Operadores incrementais e decrementais

- Os operadores incrementais e decrementais têm a função de aumentar ou diminuir exatamente em 1 o valor de uma variável. Eles podem ser pré ou pós incremental e pré ou pós decremental. Veja os conceitos de cada um deles e um exemplo prático a seguir:

Operador Aritmético	Descrição
Pré incremental (+) ou prefixo	Se o sinal for colocado antes da variável, primeiramente será somado o valor 1 para esta variável, continuando em seguida a resolução da expressão.
Pós incremental (+) ou sufixo	Se o sinal for colocado após a variável, é resolvido primeiro a expressão, seja ela adição, subtração, multiplicação ou qualquer outra, para em seguida ser adicionado o valor 1 à variável.
Pré incremental (--) ou prefixo	Se o sinal for colocado antes da variável, primeiramente será subtraído o valor 1 para esta variável, continuando em seguida a resolução da expressão.
Pós incremental (--) ou sufixo	Se o sinal for colocado após a variável, é resolvido primeiro a expressão, seja ela adição, subtração, multiplicação ou qualquer outra, para em seguida ser subtraído o valor 1 à variável.

Atribuição

- Em Ciência da Computação o comando de atribuição define ou redefine o valor armazenado no local de armazenamento indicado por um nome de variável.
- Na maioria das linguagens de programação imperativas o comando de atribuição é uma das declarações básicas.
- A instrução de atribuição muitas vezes permite que o mesmo nome de variável possa conter valores diferentes em momentos diferentes durante a execução do programa.
- Exemplo:

```
num = 10;
x = 24.45;
y = 754.2;
l = 'a';
resp = false;
nome = "Pedro";
```

Operadores Aritméticos Reduzidos

- Esses operadores são usados para compor uma operação aritmética e uma atribuição, conforme é descrito na tabela abaixo

Operador Aritmético	Descrição
+ =	mais igual
- =	menos igual
* =	vezes igual
/ =	dividido igual
% =	módulo igual

Estrutura de um Programa C#

- Antes de escrevermos um programa na linguagem C# é necessário que conheçamos sua estrutura mínima de operação e definição de código de programa, pois essa linguagem é do tipo case-sensitive. Isso significa que a linguagem diferencia caracteres maiúsculos de caracteres minúsculos, sendo necessário muita atenção e cuidado na codificação dos programas.
-
- A linguagem de programação se comunica com o computador segundo um formato sintático básico e próprio.

- As instruções de código podem ser formadas por um ou mais comandos, escritos em uma ou mais linhas.

- O final de uma instrução é indicado com o uso de um ponto e vírgula (;).

Estrutura de um Programa C#

- Para um programa C# poder executar, é necessário criar uma estrutura de comandos.
- Esta estrutura é chamada de ***template*** em alguns compiladores tais como Microsoft Visual C#, Dcoder (on-line) e repl.it (on-line):

```
using System;
class MainClass
{
  public static void Main (string[] args)
  {

  }
}
```

Estrutura de um Programa C#

- O comando using faz a chamada de um namespace específico e que cada namespace por si só é um conjunto de grupos lógicos de recursos relacionados. O namespace System é o principal e sempre deve ser incorporado ao seu programa C#.

- Todo programa C# deve ser escrito dentro de uma estrutura chamada de class. Em uma classe podemos ter atributos e métodos. Neste curso não iremos abordar a class e suas estruturas.

- O método Main é o ponto de entrada de um aplicativo C#. Quando o aplicativo é iniciado, o Main método é o primeiro método que é invocado. O estudo mais detalhado dos métodos está no Capítulo 6 desta disciplina.

Estrutura de um Programa C#

```
using System;
class MainClass
{
  public static void Main (string[] args)
  {

  }
}
```

Abre chaves do Main: significa início do programa

- O programa C# deve ser escrito neste local: entre as chaves do Main

Fecha chaves do Main: significa final do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

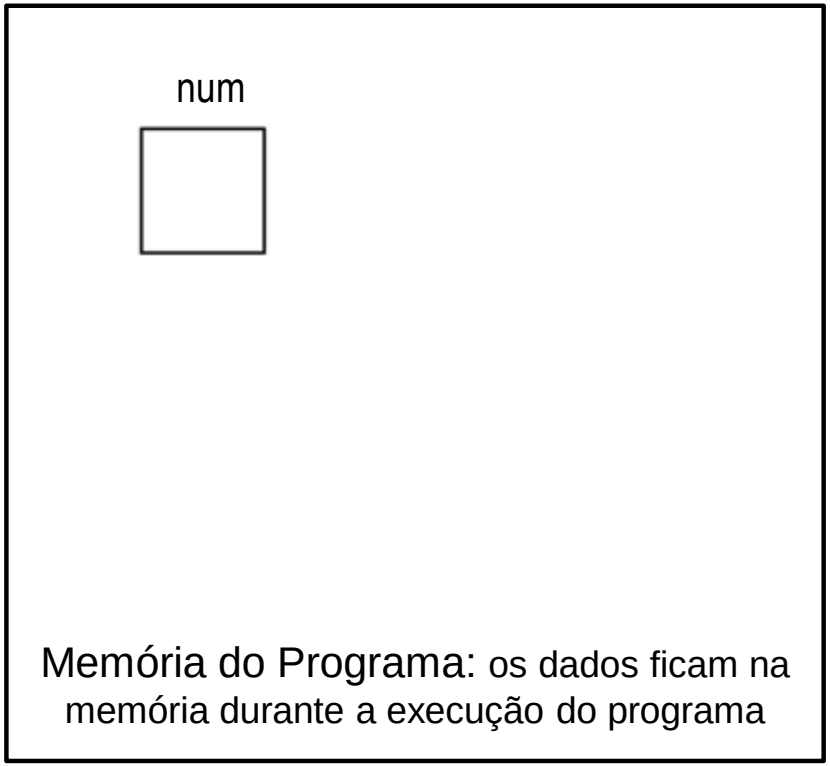
Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

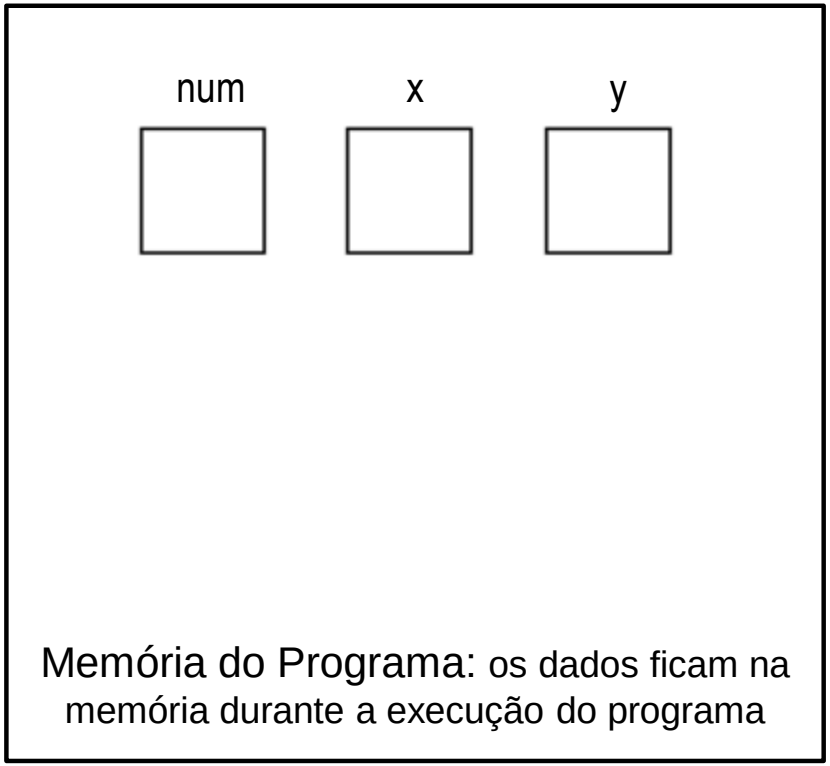
```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

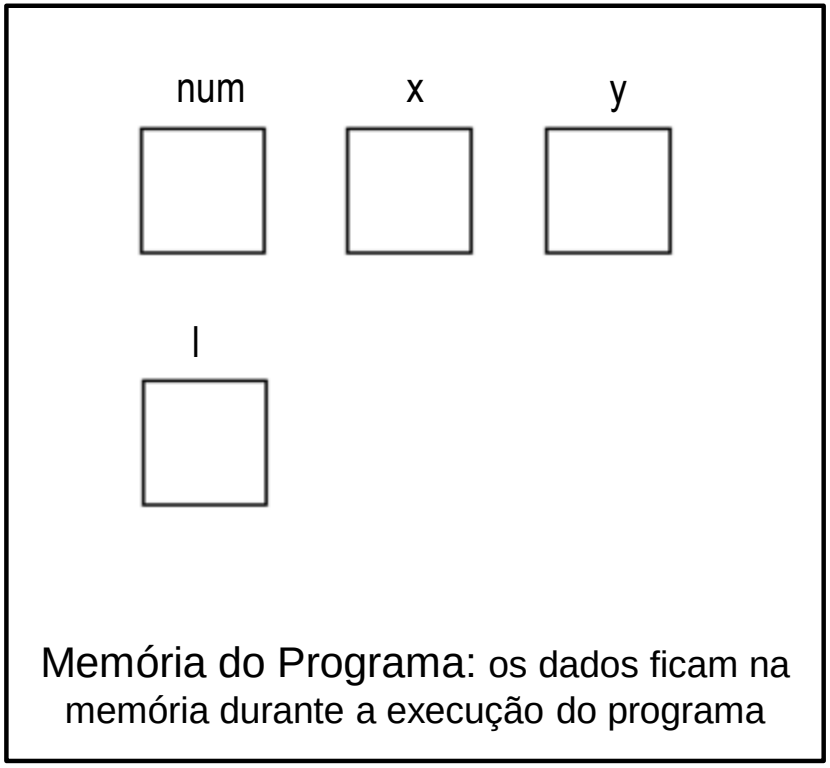
```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

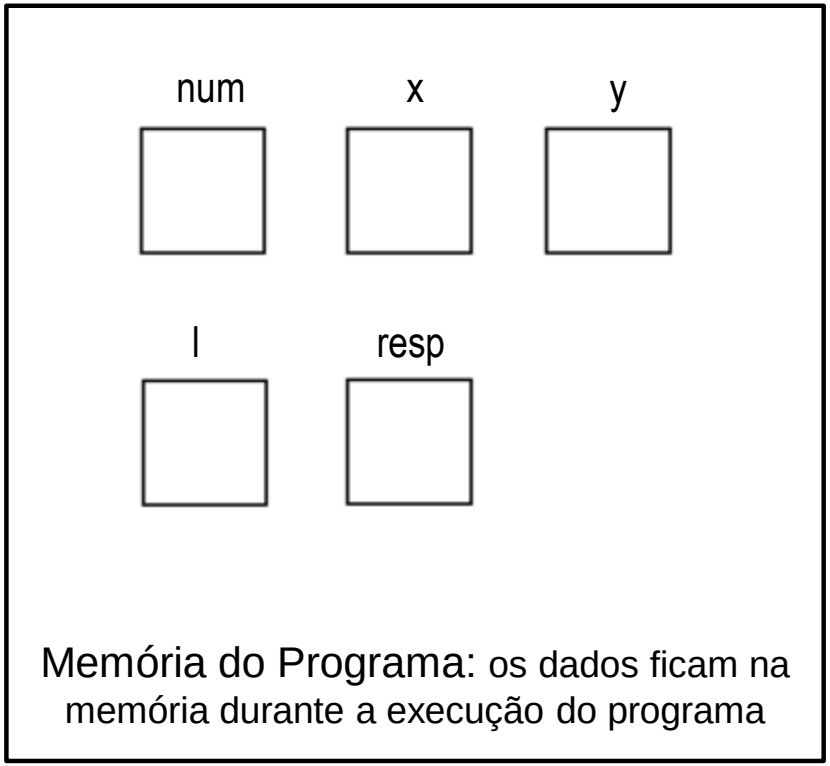
```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

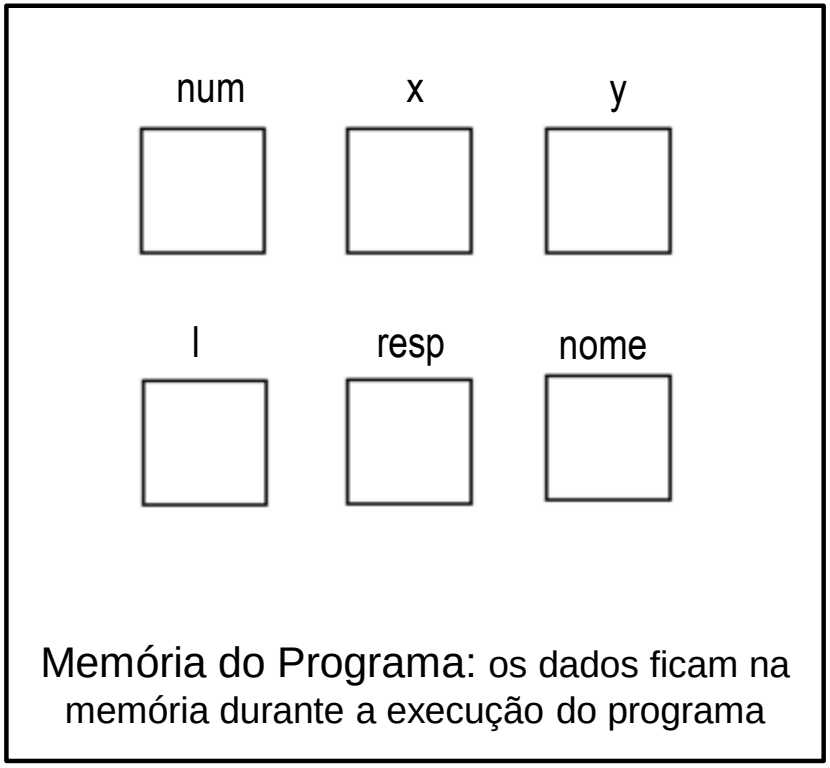
```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

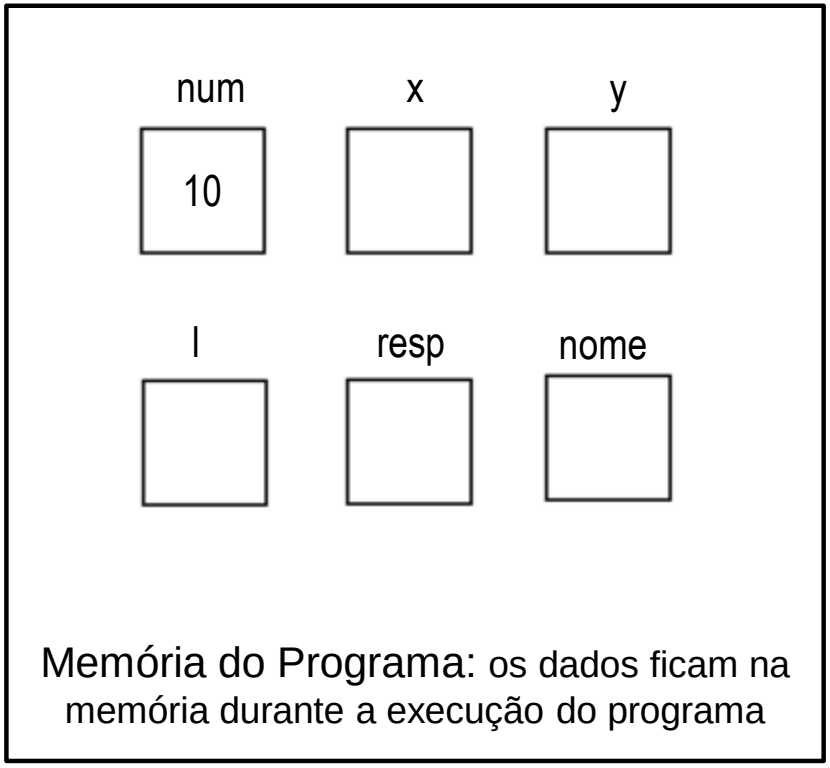


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

num recebe 10

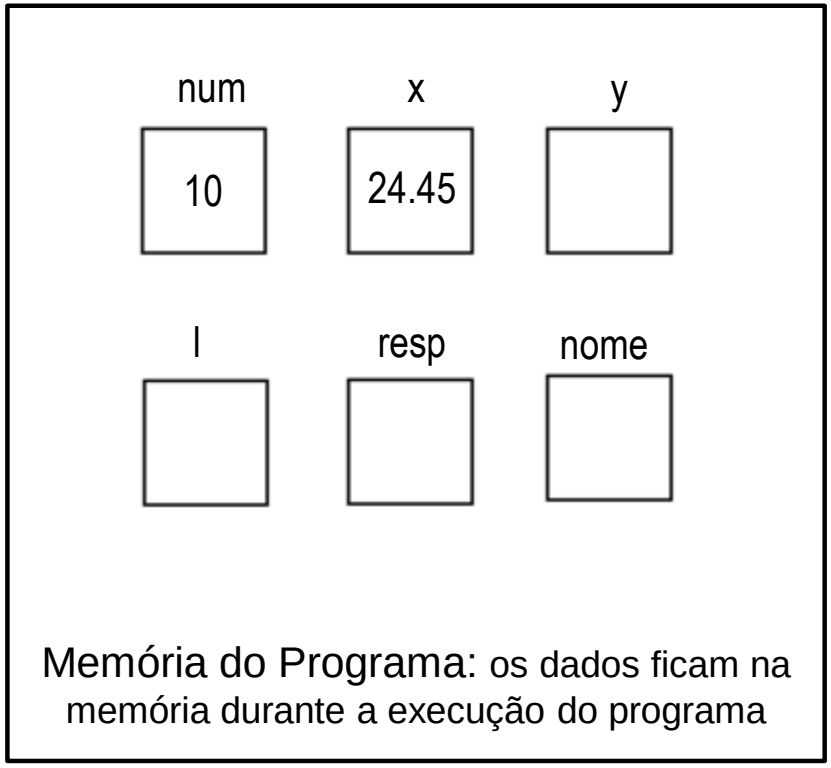


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

x recebe 24,45

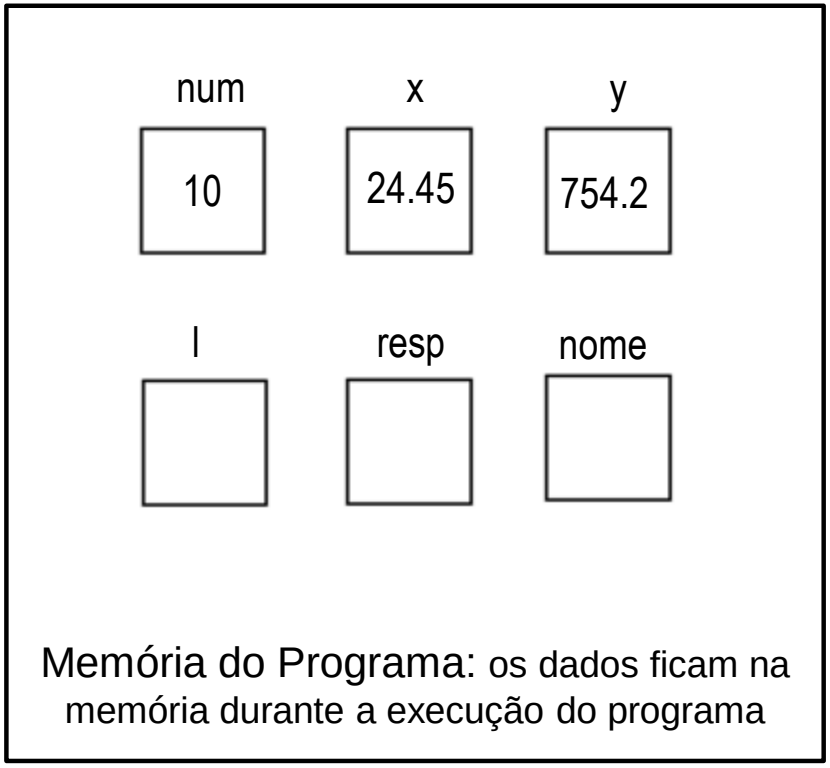


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

y recebe 754,2

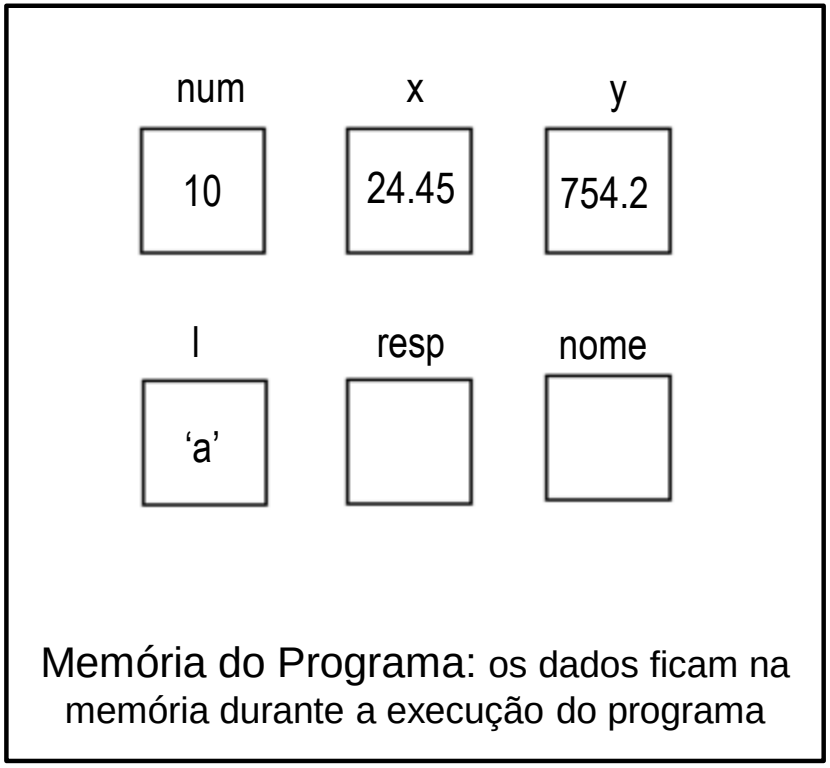


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

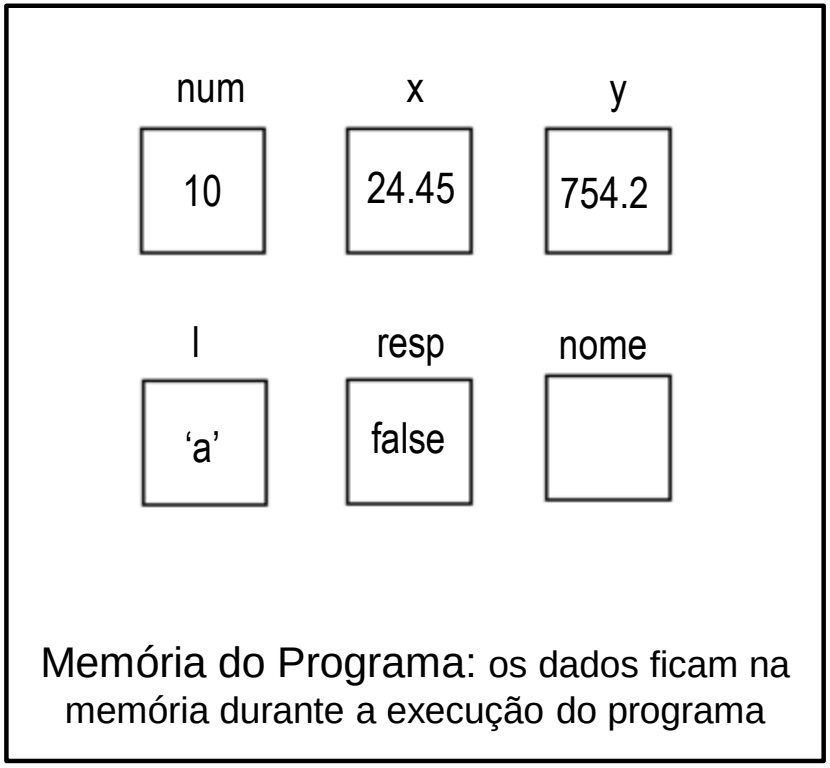
l recebe 754,2



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;  resp recebe false
    nome = "Pedro";
  }
}
```

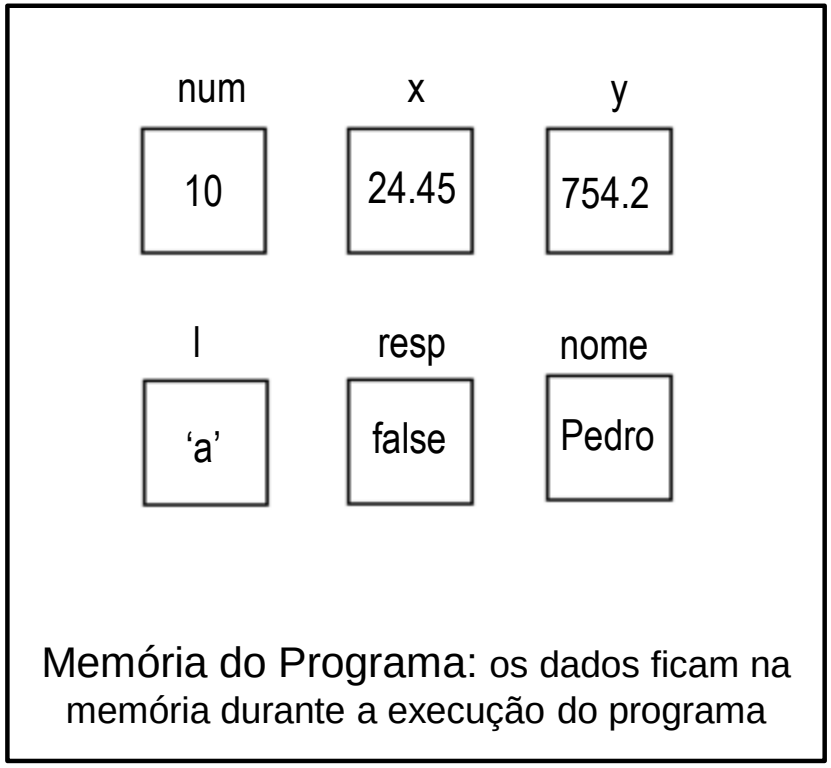


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

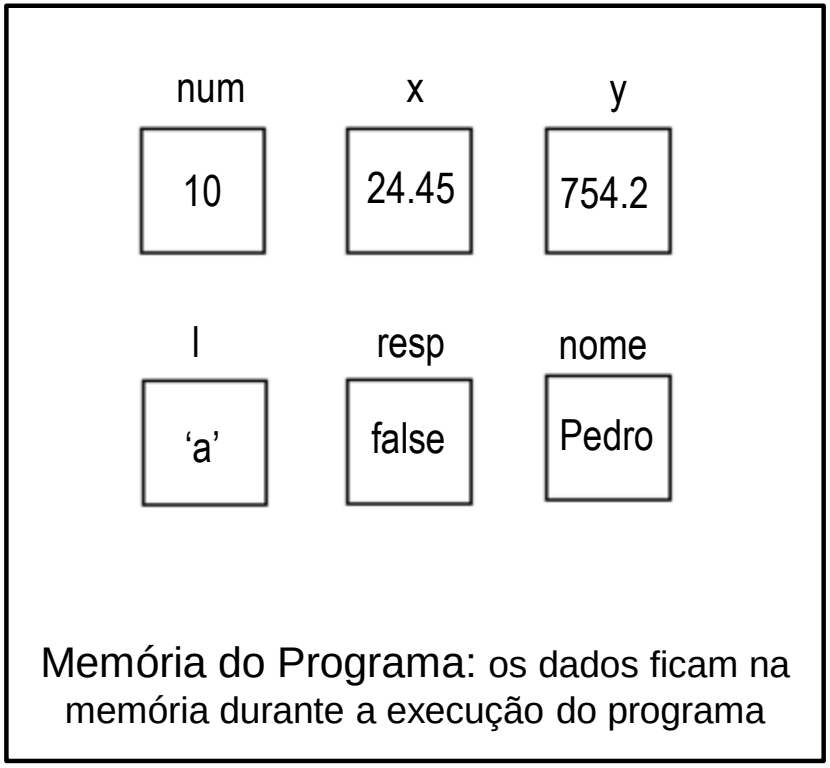
nome recebe “Pedro”



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```



Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

Este programa não mostra nenhum resultado ao ser executado.

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

Este programa não mostra nenhum resultado ao ser executado.

Por que ?

Exemplo de um Programa C#

```
using System;
class MainClass {
  public static void Main (string[] args)
  {
    int num;
    double x, y;
    char l;
    bool resp;
    string nome;

    num = 10;
    x = 24.45;
    y = 754.2;
    l = 'a';
    resp = false;
    nome = "Pedro";
  }
}
```

Este programa não mostra nenhum resultado ao ser executado.

Por que ?

Por que não tem instrução de saída da dados na tela

Programas Sequenciais

- Um programa sequencial é um programa que escrevemos a partir de três ações básicas e necessárias que um programa deve executar em um computador, que são:
  - a entrada dos dados,
  - o processamento dos dados e
  - a saída dos dados processados (informação).
- Apesar de ser essa a forma mais simples de escrevermos um programa, ela é a mais importante de todo nosso aprendizado, pois é por meio dela que escreveremos as interfaces de entrada e saída com os usuários dos nossos programas.

Comandos de Entrada e Saída de Dados

- O Console representa a entrada, as saídas, e os fluxos de erro padrão para aplicações escritas para a console.
- Comandos (métodos) para entrada de dados:
  - Console.ReadLine() – efetua entrada de dado na variável indicada;
  - Console.ReadKey() – efetua uma pausa e aguarda que alguma tecla seja acionada no teclado (útil quando utilizar MS VisualStudio)
- Comandos (métodos) para saída de dados:
  - Console.Write() – efetua a saída de algum dado no monitor de vídeo mantendo o cursor na mesma linha;
  - Console.WriteLine() – efetua a saída de dados no monitor de vídeo avançando o cursor para a próxima linha;

Comandos de Entrada e Saída de Dados

```
public static void Main (string[] args)
{
    Console.Write("Olá ");
    Console.WriteLine("mundo !");
    Console.Write("Entre com seu nome: ");
    String nome = Console.ReadLine();
    Console.Write("Bom dia, ");
    Console.Write(nome);
    Console.WriteLine("!");
}
```

Memória do Programa: os dados ficam na memória durante a execução do programa

```
Olá
```

Comandos de Entrada e Saída de Dados

```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```

Memória do Programa: os dados ficam na memória durante a execução do programa

```
Olá mundo !
```

Comandos de Entrada e Saída de Dados

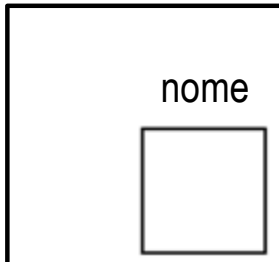
```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```

Memória do Programa: os dados ficam na memória durante a execução do programa

```
Olá mundo !
Entre com seu nome:
```

Comandos de Entrada e Saída de Dados

```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```



Memória do Programa: os dados ficam na memória durante a execução do programa

```
Olá mundo !
Entre com seu nome:
```

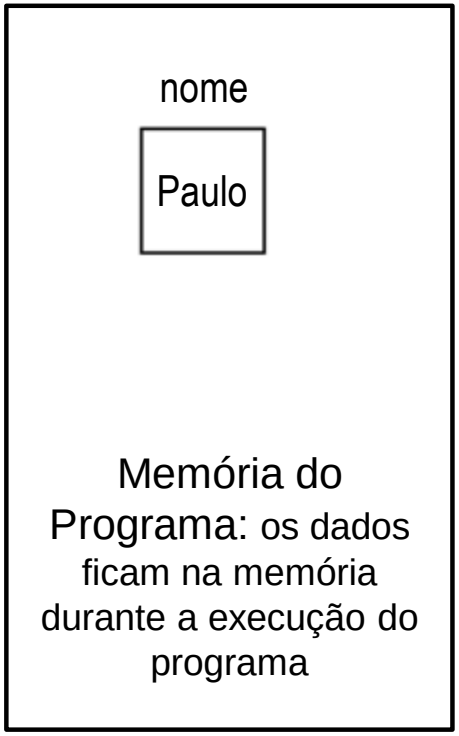
Comandos de Entrada e Saída de Dados

```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```


```
Olá mundo !
Entre com seu nome: Paulo
```

Comandos de Entrada e Saída de Dados

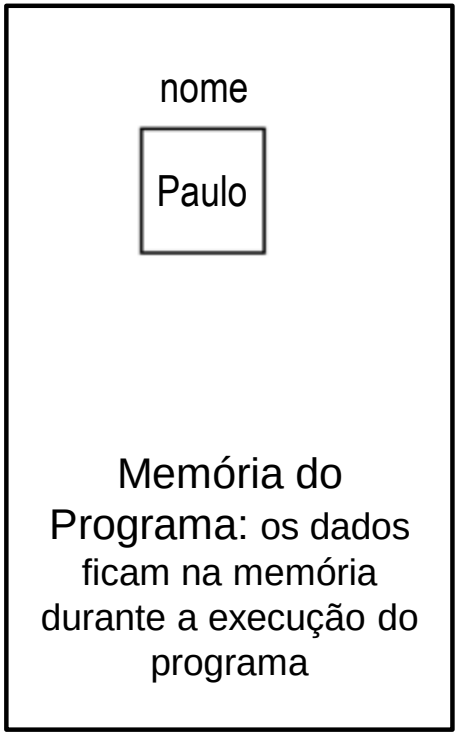
```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```




```
Olá mundo !
Entre com seu nome: Paulo
```

Comandos de Entrada e Saída de Dados

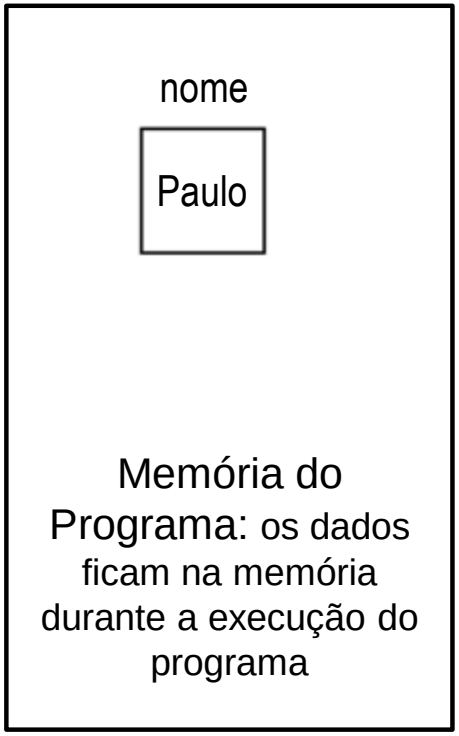
```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```




```
Olá mundo !
Entre com seu nome: Paulo
Bom dia,
```

Comandos de Entrada e Saída de Dados

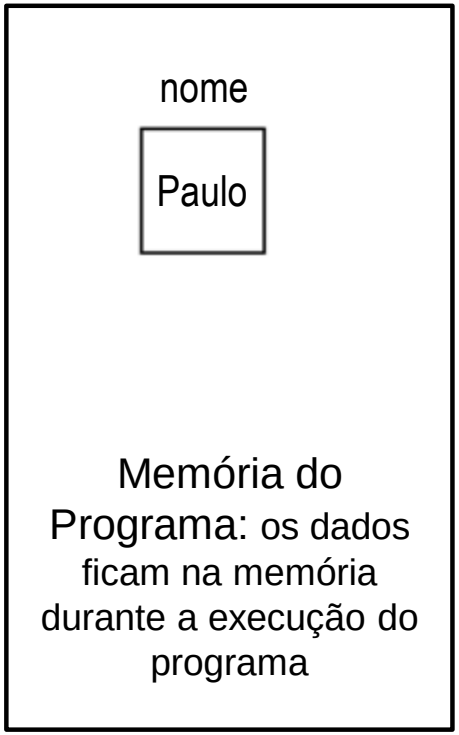
```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```




```
Olá mundo !
Entre com seu nome: Paulo
Bom dia, Paulo
```

Comandos de Entrada e Saída de Dados

```
public static void Main (string[] args)
{
  Console.Write("Olá ");
  Console.WriteLine("mundo !");
  Console.Write("Entre com seu nome: ");
  String nome = Console.ReadLine();
  Console.Write("Bom dia, ");
  Console.Write(nome);
  Console.WriteLine("!");
}
```




```
Olá mundo !
Entre com seu nome: Paulo
Bom dia, Paulo!
```

Conversão de Tipos de Dados

- O método Parse() converte uma string em um número de acordo com seu tipo de dado especificado.
- Exemplo de conversão de tipo:

```
public static void Main (string[] args)
{
  double num1;
  num1 = double.Parse("123.5");

  int num2 = int.Parse("654");
}
```

Exemplo

Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

Exemplo

Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

Vamos precisar de 2 variáveis, num1 e num2, para receber cada um dos 2 números

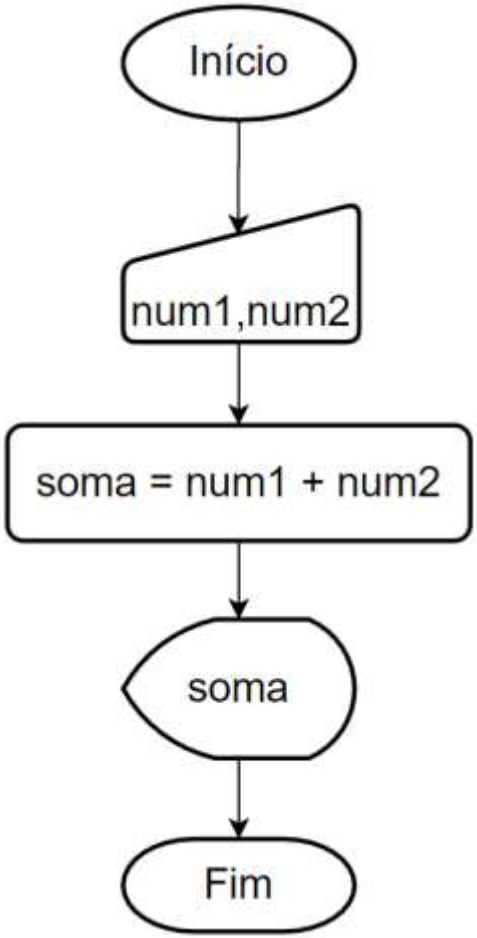
Vamos precisar de 1 variável para guardar e mostrar a soma dos 2 números num1 e num2

Exemplo

- Programa C#:

```
public static void Main (string[] args)
{
  double num1, num2, soma;
  num1 = double.Parse(Console.ReadLine());
  num2 = double.Parse(Console.ReadLine());
  soma = num1 + num2;
  Console.WriteLine (soma);
}
```

Fluxograma:

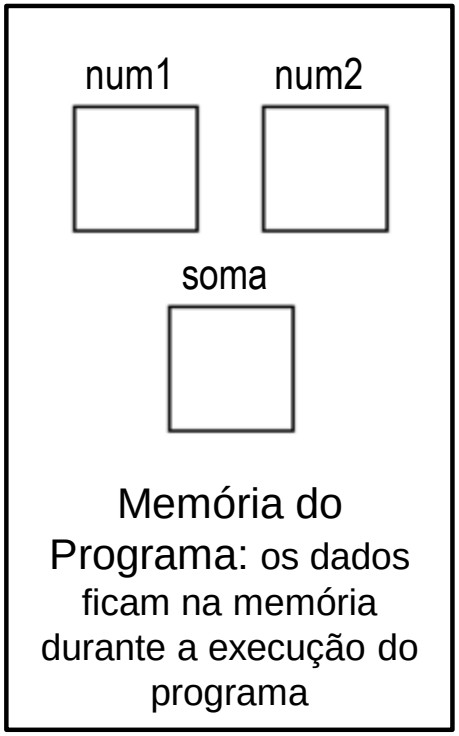


Exemplo

Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

```
public static void Main (string[] args)
{
  double num1, num2, soma;
  num1 = double.Parse(Console.ReadLine());
  num2 = double.Parse(Console.ReadLine());
  soma = num1 + num2;
  Console.WriteLine (soma);
}
```

Console: (Tela)

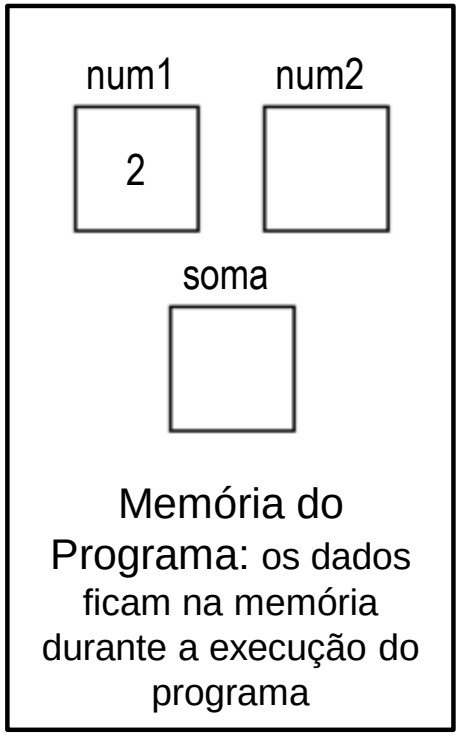


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo

Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

```
public static void Main (string[] args)
{
  double num1, num2, soma;
  num1 = double.Parse(Console.ReadLine());
  num2 = double.Parse(Console.ReadLine());
  soma = num1 + num2;
  Console.WriteLine (soma);
}
```

Memória do Programa: os dados ficam na memória durante a execução do programa

Console: (Tela)

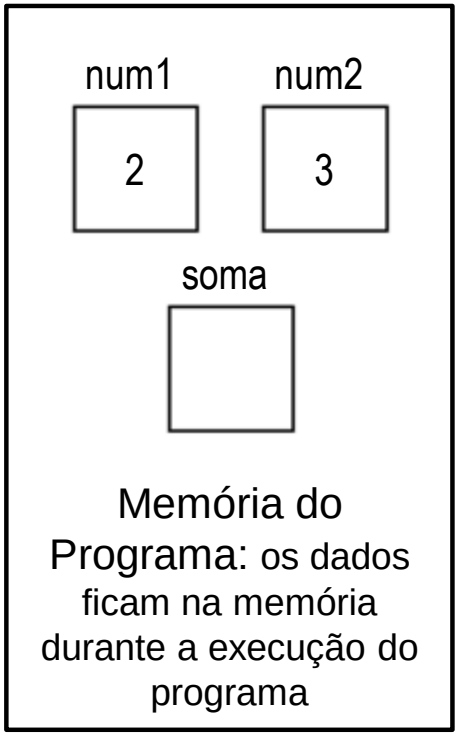
Exemplo

Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

```
public static void Main (string[] args)
{
  double num1, num2, soma;
  num1 = double.Parse(Console.ReadLine());
  num2 = double.Parse(Console.ReadLine());
  soma = num1 + num2;
  Console.WriteLine (soma);
}
```

Console: (Tela)

```
2
3
```



Memória do Programa: os dados ficam na memória durante a execução do programa

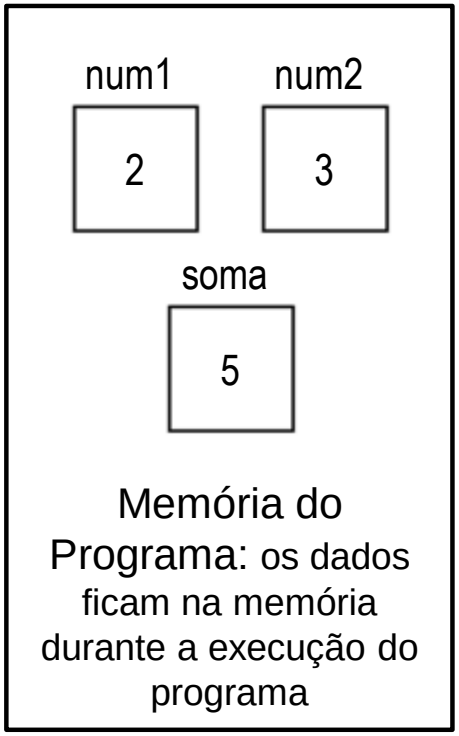
Exemplo

Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

```
public static void Main (string[] args)
{
  double num1, num2, soma;
  num1 = double.Parse(Console.ReadLine());
  num2 = double.Parse(Console.ReadLine());
  soma = num1 + num2;
  Console.WriteLine (soma);
}
```

Console: (Tela)

```
2
3
```

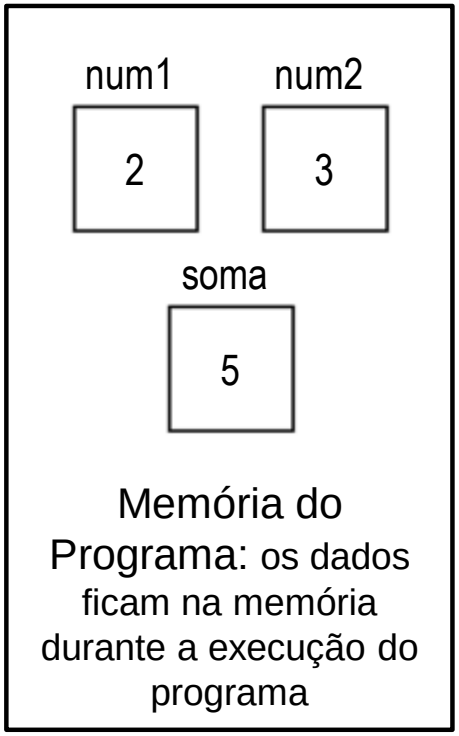


Memória do Programa: os dados ficam na memória durante a execução do programa

Exemplo

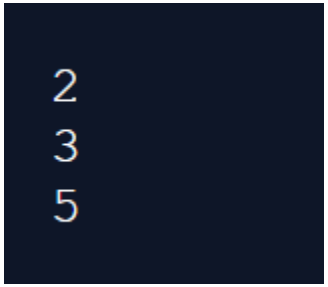
Programa que lê 2 números digitados pelo usuário e mostra a soma deles.

```
public static void Main (string[] args)
{
  double num1, num2, soma;
  num1 = double.Parse(Console.ReadLine());
  num2 = double.Parse(Console.ReadLine());
  soma = num1 + num2;
  Console.WriteLine (soma);
}
```

Memória do Programa: os dados ficam na memória durante a execução do programa

Console: (Tela)



Exercícios

1. Fazer programa para calcular a área de um quadrado.
2. Fazer programa para calcular a área de um trapézio.
3. Fazer programa que o usuário digita o salário atual e o percentual do aumento salarial e no final programa mostra o novo salário.
4. Fazer programa que calcula a tensão elétrica (em Volts) onde o usuário entra com o valor da corrente elétrica (em Amperes) e o valor da resistência (em Ohms).

Conceitos desta aula

- Tipos primitivos de dados (numérico, carácter ou lógico);
- Variável / Constante;
- Memória RAM / Volátil
- Operadores Aritméticos;
- Expressões Aritméticas;
- “Texto entre aspas”;
- /* // Comentários */;
- Incremento/ decremento;
- Case-sensitive;
- Final de instrução (;);
- Namespace;
- Classe (atributos e métodos);
- Console.ReadLine() / Console.ReadKey();
- Console.Write() / Console.WriteLine().