

Programação de Computadores

VARIÁVEIS, CONSTANTES E ATRIBUIÇÃO

Introdução

- Um programa é uma **sequência de instruções**
 - A execução se inicia pela função principal (**main**)
 - Só pode haver **uma função principal** em cada projeto

Definição da função

```
int main()  
{  
  
    instruções  
  
    return 0;  
}
```

Cabeçalho da função

Corpo da função

Introdução

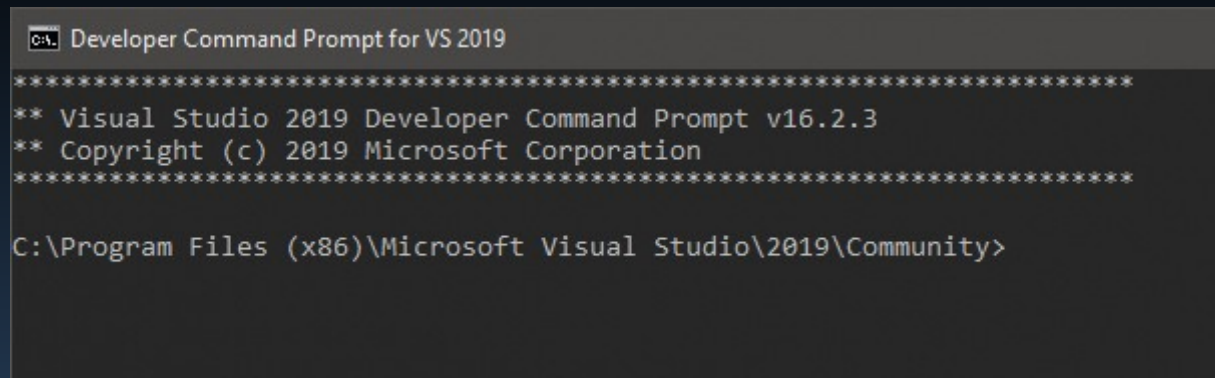
- A biblioteca `iostream` nos permite usar `cout` e `endl` para fazer a saída de dados na tela

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Oi Mundo!" << endl;
    return 0;
}
```

Introdução

- Um programa pode ser **compilado e executado**:
 - Usando o ambiente integrado do Visual Studio
 - Botão "**Depurador Local do Windows**" (F5)
 - Através da linha de comando
 - Usando o **Prompt de Comando do Desenvolvedor**

A screenshot of the 'Developer Command Prompt for VS 2019' window. The title bar is dark gray with a small icon and the text 'Developer Command Prompt for VS 2019'. The main area is black with white text. It displays a welcome message: '*****', '** Visual Studio 2019 Developer Command Prompt v16.2.3', '** Copyright (c) 2019 Microsoft Corporation', and '*****'. Below this, the current directory is shown as 'C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>'.

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>
```

Instruções Básicas

```
// cenouras.cpp - processamento de comida
#include <iostream>
using namespace std;

int main()
{
    int cenouras;           // declara uma variável inteira
    cenouras = 25;          // atribui um valor a uma variável

    cout << "Eu tenho ";
    cout << cenouras;        // exibe o valor da variável
    cout << " cenouras.";
    cout << endl;

    cenouras = cenouras - 1; // modifica a variável
    cout << "Agora eu tenho " << cenouras << " cenouras." << endl;

    return 0;
}
```

Constantes *versus* Variáveis

- Uma constante tem **valor fixo e inalterável**
 - Inalterável durante a execução do programa
 - Elas podem ser de tipo:
 - **Numéricas**: números na base 2, 8, 10 ou 16
 - **Caractere**: letras e símbolos
 - **String**: representam texto
 - **Booleanas**: valores verdade **true** e **false**

Constantes *versus* Variáveis

- Constantes para números
 - **Decimal**: números na base 10 são escritos de forma direta.
Ex.: 25, 403, 8390
 - **Hexadecimal**: números na base 16 devem ser precedidos por 0x (representação comum para endereços de memória).
Ex.: 0x19, 0x193, 0x20C6
 - **Octal**: números na base 8 devem ser precedidos por um 0 (zero).
Ex.: 031, 0623, 020306
 - **Binário**: números na base 2 devem ser precedidos por 0b
Ex.: 0b11001, 0b110010011, 0b10000011000110

Constantes *versus* Variáveis

- Constantes para letras, símbolos e texto
 - **Caractere**: deve ser escrito entre aspas simples.
Ex.: 'a', 'b', 'B', '*', '?'
 - **String**: um conjunto de caracteres deve ficar entre aspas duplas.
Ex.: "primeiro programa", "oi", "a"
- Existe uma diferença entre as constantes 'a' e "a":
 - 'a' é um caractere
 - "a" é um conjunto de caracteres

Constantes *versus* Variáveis

```
// constvar.cpp - exemplifica o uso de constantes
#include <iostream>
using namespace std;

int main()
{
    cout << "A letra " << 'm';
    cout << " tem " << 3 << " pernas." << endl;

    char ch = 'M';           // tipo caractere
    int num = 0x1E;          // tipo inteiro

    cout << "Dez letras " << ch;
    cout << " têm " << num << " pernas." << endl;

    return 0;
}
```

Constantes *versus* Variáveis

- Uma **variável** permite ao programador **guardar informações**
 - Podem ser alteradas durante a execução do programa



Informações guardadas em variáveis
são **armazenadas na memória**
do computador

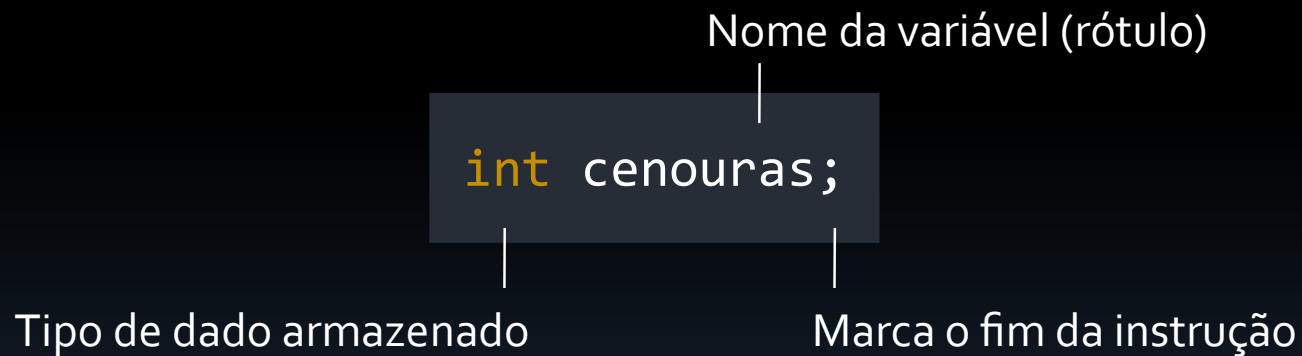
Declaração de Variáveis

- Para **guardar informação na memória** é preciso indicar:
 - O endereço de armazenamento
 - A quantidade de memória necessária



Declaração de Variáveis

- A forma mais simples de guardar informações é através de uma **declaração de variável**



- A declaração fornece:
 - O tipo da informação (**quantidade de memória**)
 - Um rótulo (**endereço de memória**)

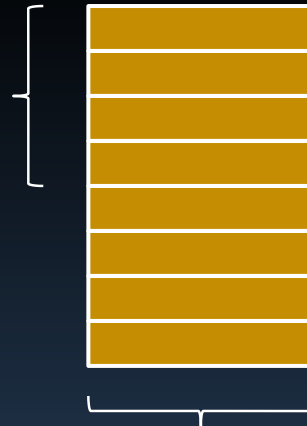
Declaração de Variáveis

Tipo de dado armazenado

Nome da variável

```
int cenouras;
```

int



0xCB20 = cenouras

0xCB21

0xCB22

0xCB23

0xCB24

0xCB25

0xCB26

0xCB27

Endereços de memória

Dados

Declaração de Variáveis

- Ao declarar a variável **cenouras**:
 - Será alocada memória suficiente para guardar um **inteiro**
 - O nome **cenouras** vai fazer referência a um **endereço de memória**
 - **cenouras** é uma **variável** porque o seu valor pode ser alterado

```
int cenouras;
```

- Em C++ **todas as variáveis devem ser declaradas**

Declaração de Variáveis

- Por que **as variáveis devem ser declaradas?**
 - Em BASIC as variáveis são criadas automaticamente

Programa válido em BASIC

```
cenouras = 34;  
...  
cenoura = cenouras + 1;  
PRINT cenouras
```

- A criação automática de variáveis pode gerar **erros difíceis de achar**

Declaração de Variáveis

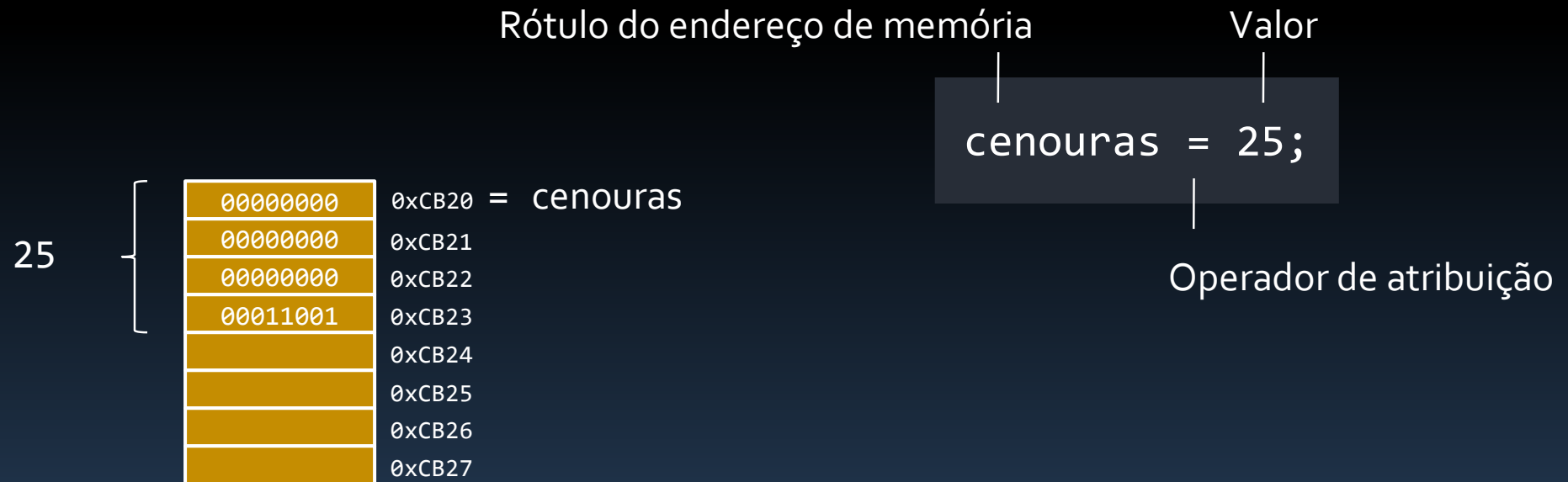
- A declaração pode ser feita em qualquer ponto do programa, sempre **antes do primeiro uso da variável**

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Eu tenho vegetais para vender.\n";
    int cenouras, beterrabas, alfaces;
    cenouras = 25;
    beterraba = 10;
    alfaces = 18;
    ...
}
```


Atribuição de Valor

- Uma **instrução de atribuição** coloca um valor em um endereço de memória



Atribuição de Valor

- Em linguagem C/C++ é permitido **usar o operador de atribuição de forma serial**

```
int main()
{
    int cenouras;    // declara uma variável
    int alfaces;     // declara outra variável
    int tomates;     // declara mais uma variável

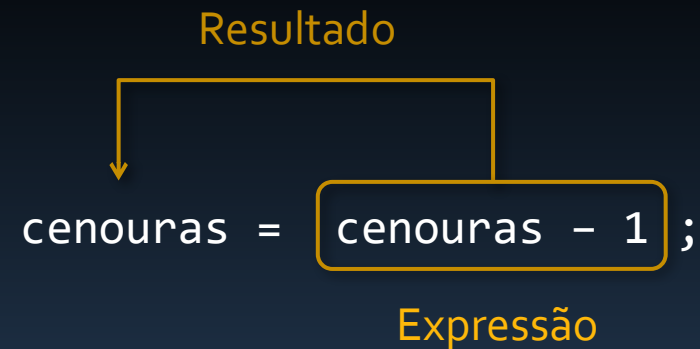
    // atribui 25 a todas as variáveis
    cenouras = alfaces = tomates = 25;
    ...
}
```

Atribuição de Valor

- A segunda instrução de atribuição em `cenouras.cpp` mostra que o valor de uma variável pode ser alterado

```
cenouras = cenouras - 1;    // modifica a variável
```

- **Primeiro** a expressão matemática é avaliada
- **Depois** o resultado é atribuído a variável `cenouras`



Atribuição de Valor

- Atribuições **sobrescrevem os valores antigos** daquela posição de memória

```
#include <iostream>
int main(void)
{
    int a, b;
    a = 5;
    b = 2;      // linha 7
    b = a;      // linha 8
    a = b;      // linha 9

    std::cout << b << a;
}
```

Exibindo Variáveis

- `cout` é capaz de **exibir o conteúdo de variáveis**

Periférico de saída

Variável inteira

```
cout << cenouras;
```

- `cout` exibe tanto texto como valores inteiros

```
cout << cenouras;  
cout << "cenouras";
```

cout *versus* printf

- cout é inteligente o suficiente para **identificar o tipo de dado e convertê-lo** para a saída em formato texto

```
// função de saída de dados da linguagem C++  
cout << "Exibindo uma string: " << "25";  
cout << "Exibindo um inteiro: " << 25;  
cout << "Exibindo uma string: " << "cenouras";  
cout << "Exibindo um inteiro: " << cenouras;
```

```
// função de saída de dados da linguagem C  
printf("Exibindo uma string: %s", "cenouras");  
printf("Exibindo um inteiro: %i", cenouras);
```

Entrada de Dados em C++

```
// lerinfos.cpp - entrada e saída de dados
#include <iostream>
using namespace std;

int main()
{
    int cenouras;

    cout << "Quantas cenouras você tem?" << endl;
    cin >> cenouras; // entrada de dados em C++

    cout << "Aqui estão mais duas. ";
    cenouras = cenouras + 2;

    // concatena a saída
    cout << "Agora você tem " << cenouras << " cenouras." << endl;

    return 0;
}
```

Entrada de Dados com cin

- O **valor digitado no teclado** é atribuído à variável `cenouras`

A entrada padrão é o teclado

Variável tipo inteiro

```
cin >> cenouras;
```

Operador de extração

- Para usar `cin` é necessário incluir **`iostream`**

Concatenando Leituras

- `cin` permite concatenar múltiplas leituras em uma única instrução

```
// leituras em sequência  
cin >> cenouras >> bananas >> abacaxi;
```

- Não há diferença em usar vários `cin`'s separados:

```
// leituras separadas  
cin >> cenouras;  
cin >> bananas;  
cin >> abacaxi;
```

Buffer do Teclado

- A entrada de texto com **cin utiliza um buffer**
 - Agrupa os dados até o pressionamento do ENTER
 - Permite uma leitura mais eficiente

00110011	00111000	00101110	00110101	00100000	00110001	00111001	00101110	00110010	00001101
'3'	'8'	' '	'5'	' '	'1'	'9'	' '	'2'	'\n'

```
// variáveis recebem valores apenas após pressionamento do ENTER
cin >> x;
cin >> y;
cin >> z;
cin >> w;
```

Concatenando Escritas

- cout também permite concatenar múltiplos elementos

```
// concatena a saída
cout << "Agora você tem " << cenouras << " cenouras." << endl;
```

- Não há diferença entre espaço, tabulação e salto de linha:

```
// concatena a saída
cout << "Agora você tem "
    << cenouras
    << " cenouras."
    << endl;
```

```
// mesmo resultado
cout << "Agora você tem ";
cout << cenouras;
cout << " cenouras.";
cout << endl;
```

Resumo

- Dados podem aparecer em um programa na forma de **constantes** ou de **variáveis**
 - Variáveis podem ter seu conteúdo modificado †
 - Constantes são valores inalteráveis †
- Variáveis podem ser modificadas através:
 - De atribuições de valores
 - Com a instrução de entrada de dados (cin)
 - É preciso incluir **iostream** para usar o cin