Linguagem de Programação C: Conceitos e Fundamentos

Disciplina: Algoritmos e Programação
Curso: Engenharia de Computação

Professora: Mariza Miola Dosciatti
mariza@utfpr.edu.br

Objetivos

- Entender o conceito de programa e como um algoritmo é representado em uma linguagem de programação.
- Compreender a ideia de função e saber utilizar os itens fundamentais de uma linguagem de programação:
    - Função main()
    - Variáveis
    - Tipos de dados
- Entender as funções básicas de entrada e saída.
- Conhecer as palavras-chave da linguagem C.
- Conhecer os operadores básicos utilizados na linguagem C.
- Conscientizar-se do formalismo e da estrutura de um programa em linguagem C.

- Itens da ementa (Plano de Ensino):
  - Representação de dados.
  - Tipos de dados: conceitos, formas de representação.
  - Elementos básicos de uma linguagem de programação estruturada.

Sumário

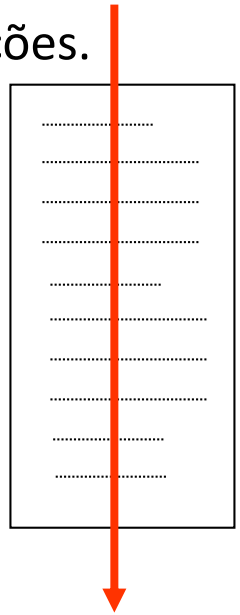
1. Programa (cont.)

- **Um programa** é basicamente **composto por**:
  - **Entrada**
    - Obtenção de dados para o programa trabalhar (receber dados do usuário ou de outras fontes).
  - **Processamento**
    - Manipulação de constantes e de variáveis.
    - Resolução de expressões matemáticas.
    - Execução de instruções sequenciais, de decisão e de repetição.
    - Manipulação de dados em bases de dados e arquivos.
    - ...
  - **Saída**
    - Informar dados de processamento (para o usuário, armazená-los, transmiti-los, acionar dispositivos...).

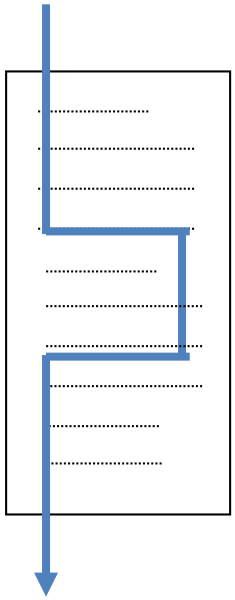
1. Programa (cont.)

- **Entrada**
  - Variáveis e constantes.
- **Processamento**
  - Instruções.

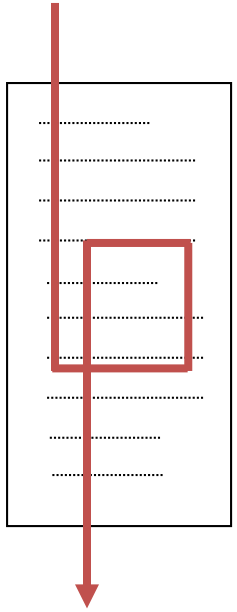
fluxo de execução sequencial



fluxo de execução com desvio



fluxo de execução com repetição



- **Saída**
  - Texto, conteúdo de variáveis e formatação.

1. Programa (cont.)

- **Método para a construção de um programa computacional:**
  1. Entender o problema.
  2. Retirar do problema as entradas de dados necessários.
  3. Definir as saídas que o programa deve fornecer.
  4. Determinar o que deve ser feito para transformar as entradas nas saídas (o algoritmo), resolver o problema.
     - Determinar o tipo de dado a ser manipulado, definindo as variáveis e as constantes necessárias.
     - Definir os cálculos, fórmulas e outros.
     - Definir as instruções e as estruturas de decisão e de repetição necessárias.
  5. Apresentar os resultados.
  6. Verificar se as instruções definidas resolvem o problema da maneira esperada.
     - Teste de mesa, por exemplo.

1. Programa (cont.)

- **Considerações:**
  - No início do programa colocar o enunciado do problema.
  - Comentar as partes mais complexas ou relevantes do programa.
  - Utilizar espaços e linhas em branco para melhorar a legibilidade do programa, mas sem abusar dos mesmos.
  - Escolher nomes representativos para os identificadores.
  - Uma instrução (comando) por linha é suficiente.
  - Utilizar parênteses para aumentar a legibilidade das expressões e evitar erros.
  - Utilizar indentação adequada. Indentar o código é fundamental para sua legibilidade.

Exemplo:

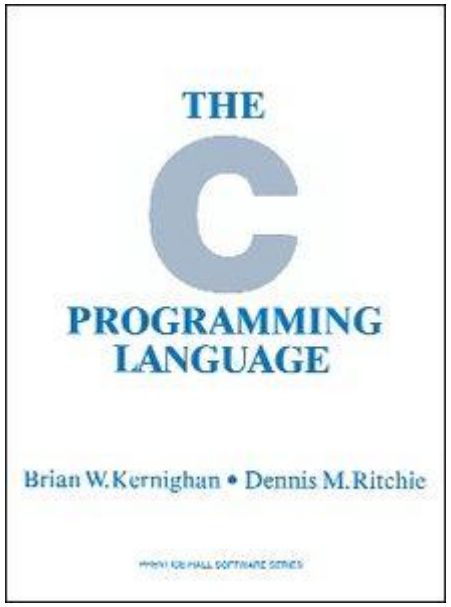
```
 1  /*Tendo como entrada dois valores inteiros, elaborar um algoritmo para calcular e
 2  mostrar a soma desses números*/
 3
 4  #include <stdio.h>
 5
 6  int main(void)
 7  {
 8      //Declarar variáveis
 9      int num1, num2, soma;
10
11      //Entrada
12      printf("Informe um valor: ");
13      scanf("%d",&num1);
14      printf("Informe outro valor: ");
15      scanf("%d",&num2);
16
17      //Processamento
18      soma = num1 + num2;
19
20      //Saída
21      printf("Soma: %d", soma);
22
23      return 0;
24  }
```

2. Linguagem C

- É uma linguagem:
  - **Compilada:** código executado diretamente pelo sistema operacional. Não há necessidades de interpretadores instalados.
  - **Estruturada:** baseada em estruturas sequenciais, de decisão e de repetição.
  - **Imperativa:** a computação é realizada com ações (instruções/comandos) que alteram o estado (variáveis) de um programa.
  - **Procedural:** uso de funções (modular).
  - **Alto nível:** próxima da linguagem natural, porém também considerada de baixo nível porque permite que o programador tenha mais controle sobre os recursos do sistema, como memória e processos. Por exemplo, em C é comum trabalhar com ponteiros e manipulação direta de memória.

2. Linguagem C (cont.)

- Padronizada pela ISO (*International Organization for Standardization*) e definida por uma norma ANSI (*American National Standards Institute*) em 1988.
- Criada por Dennis M. Ritchie em 1972, nos laboratórios Bell.
- A linguagem C não teve um sucesso imediato após a sua criação e o seu uso ficou restrito a alguns laboratórios.
- Em 1978 Dennis Ritchie e Brian Kernighan lançaram o livro *The C Programming Language* que serviu de tutorial e mudou a história da programação em C.



2. Linguagem C (cont.)

- A linguagem C é amplamente reconhecida por sua eficiência e versatilidade, e muitos sistemas importantes foram construídos com ela. Aqui estão alguns dos sistemas e softwares mais significativos desenvolvidos em C:
  - 1. Sistemas Operacionais
    - Unix, Linux e Windows:
  - 2. Compiladores e Interpretes
    - GCC (GNU Compiler Collection) e LLVM
  - 3. Software de Rede
    - Apache HTTP Server e Postfix
  - 4. Bancos de Dados
    - MySQL e SQLite
  - 5. Sistemas Embarcados e Firmware
  - 6. Aplicações Científicas e Técnicas
    - MATLAB e Blender
  - 7. Jogos
  - 8. Ferramentas de Desenvolvimento
    - Git e Make

2.1. Palavras reservadas da linguagem C

- Possui 32 palavras reservadas pelo padrão ANSI.
- Todas são escritas em letras minúsculas e não podem ser usadas como identificadores.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

2.2. Estrutura de um programa C

- Um programa em C é constituído por uma ou mais *funções.*
- Em um programa C **executável** deve existir **uma e somente uma** função com a identificação ***main*.**

- Estrutura básica de um programa em C:

```
main(void)
{

}
```

ou

```
int main(void)
{

}
```

ou

```
int main(void)
{

    return 0;
}
```

2.2. Estrutura de um programa C (cont.)

```c
 /* Programa em Linguagem C */
//biblioteca de funções de entrada e saída de dados
  #include <stdio.h>

  int main(void) //função principal

  {//marca o início do bloco de instruções

      //instruções

      printf("UTFPR");

      return 0;

 } //marca o fim do bloco de instruções e da função main
```

2.2. Estrutura de um programa C (cont.)

```c
#include <stdio.h>       //biblioteca de funções

 int main(void)   //função principal
 {
     float nota1, nota2, media;

     //entrada dos dados
     printf("Informe a primeira nota: ");
     scanf("%f",&nota1);
     printf("Informe a segunda nota: ");
     scanf("%f",&nota2);

     //processamento dos dados
     media = (nota1 + nota2)/2;

     //Saída dos dados
     printf("Media: %.2f", media);

     return 0;
 }
```

//instruções

2.2.1. Diretiva include

- **Bibliotecas** contêm funções (código que implementa funcionalidades específicas) para serem reutilizadas.
- A diretiva **#include** instrui o compilador **a ler** o conteúdo de **um arquivo** e a considerar o conteúdo desse arquivo no programa.
- O **nome do arquivo** deve estar **entre aspas** (procura-o no diretório em que está a main()) ou **entre parênteses angulares** (procura-o nos diretórios padrão de include para o compilador). É possível incluir o caminho de localização do arquivo.

```
#include  “biblioteca1.h”
#include  <stdio.h>
#include “arquivo1.c”
```

- **Recomendação:**
  - “ ” para incluir arquivos do projeto;
  - < > para incluir os arquivos de cabeçalho padrão.

2.2.2. Ponto e vírgula e Chaves

- **Ponto e vírgula:**
  - Toda a instrução ou comando em C termina obrigatoriamente com “;”

- **Chaves:**
  - Utilizadas para definir um bloco ou um conjunto de instruções relacionadas e logicamente conectadas.
  - Um bloco começa com uma “{” e termina com uma “}”

2.2.3. Bloco de comandos

- Um bloco de comandos ou conjunto de instruções pode ser colocado em qualquer lugar em que seja possível a colocação de uma única instrução.
- **Exemplo:**

```
#include <stdio.h>  //contém as funções scanf() e printf()

int main(void)
{
      int a, b;

      printf("Digite um numero: ");
      scanf("%d",&a);
      printf("Digite outro numero: ");
      scanf("%d",&b);

      if(a<b)
      {
          printf ("A Soma dos numeros eh: %d", a+b);
      }

      return 0;
 }
```

Bloco de instruções

Bloco de instruções

2.2.4. Comentários

- Comentários são anotações desconsideradas na compilação/interpretação e na execução do programa.
  // é comentado a partir da especificação **//** até o final da respectiva linha.
  /* ... */ é comentário com início e fim delimitado, pode incluir múltiplas linhas ou parte de texto dentro de uma linha.

- **Exemplos:**

```
/* O cálculo do imposto deve considerar a alíquota do ICMS. */
preco = precobruto /*preço de fábrica*/ + imposto + lucro +
despesas;

//mostrar o valor armazenado na variável preço
printf("Preço do produto é %.2f", preco); //imprimir o
preço
```

3. Elementos fundamentais da linguagem C

- Elementos utilizados para compor as instruções de um programa em C:
  - Variáveis e constantes;
  - Tipos de dados;
  - Instruções de entrada e saída;
  - Operadores aritméticos, relacionais e lógicos;
  - Estrutura sequencial;
  - Estruturas de controle decisão;
  - Estruturas de controle repetição;
  - Estruturas de dados homogêneos;
  - Estruturas de dados heterogêneos;
  - Funções.

3.1. Variáveis

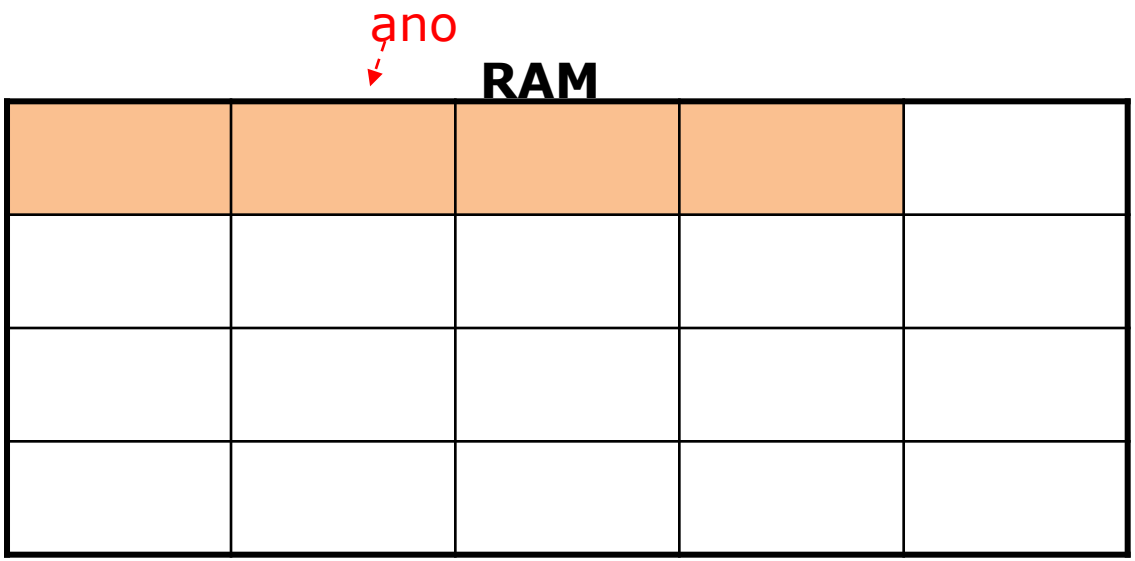
- **Variáveis** representam **valores** que podem mudar ao longo do tempo ou em execuções sucessivas de um programa.
- Na prática, uma **variável** indica (**está associada**) a uma **posição de memória** reservada, identificada por um nome (identificador), que pode ser utilizada para armazenar um valor de um tipo de dado especificado. E esse valor poder ser recuperado (lido) por meio do identificador.
- Uma **variável possui um nome** que a identifica **e um tipo** que determina o conteúdo que ela pode armazenar. Esse tipo especifica a "quantidade" necessária de células de memória para armazenar um valor desse tipo.

3.1. Variáveis (cont.)

Exemplo: Declarar uma variável inteira para armazenar um valor do tipo inteiro.

```
int ano;
```

Reserva espaço de memória suficiente para armazenar um valor inteiro.



3.1. Variáveis (cont.)

- O computador não trabalha diretamente com os identificadores (os nomes definidos para as variáveis). **A manipulação das células de memória é feita por meio dos seus endereços.**

- O **tradutor** (compilador ou interpretador) **faz uma associação** entre os identificadores e os endereços de memória reservados para as variáveis.

3.1. Variáveis (cont.)

- Todas as variáveis em C devem ser declaradas.
- Forma geral de declaração de variáveis:

```
tipo identificador_da_variável;
```

Exemplo:

```
int ano;
```

  - **identificador_da_variável** é o nome da variável, utilizado para acessar o seu conteúdo e para atribuir-lhe um valor para ser armazenado no endereço de memória reservado para a mesma.

  - **tipo** é qualquer tipo de dado válido (*char*, *int*, *float*, *double*) mais qualquer modificador válido para o respectivo tipo de dado (*long*, *short*, *signed*, *unsigned*).

3.1. Variáveis (cont.)

- **Identificadores** são nomes de variáveis, de funções e de outros elementos definidos pelo programador.
  - Podem possuir até 32 caracteres.
  - O primeiro caractere deve ser letra ou sublinhado e os caracteres subsequentes podem ser letras, números ou sublinhados (*underline*).
  - C é *case sensitive:*

    Nome, NOME, NoMe - são identificadores distintos.
  - O nome (identificador) não pode ser o nome de uma palavra reservada: *auto*, *static*, *extern*, *int*, *long*, *if*, *while*, *do*, ...

3.1. Variáveis (cont.)

- **Tipos de dados** representam os tipos de valores que podem ser manipulados por um programa.
- O tipo do dado:
  - Determina o conjunto de valores que uma variável pode armazenar.
  - Determina o espaço de memória necessário para armazenar um valor do respectivo tipo.
  - Define as operações que podem ser realizadas com as respectivas variáveis.
    - **Exemplo:** o operador % (resto) somente pode ser utilizado com valores inteiros.

3.1. Variáveis (cont.)

- Os tipos de dados na linguagem C são:
  - Elementares (básicos):
    - inteiro: *int*
    - real ou ponto flutuante: *float* e *double*
    - caractere: *char*
    - sem valor*: void*
  - Estruturados (compostos por tipos elementares):
    - Homogêneos
      - Vetores de caracteres *(string) e numéricos, unidimensionais* (*array*) *e multidimensionais (matriz)*
    - Heterogêneos
      - registro *(struct)*
      - arquivo em disco *(file)*

3.1. Variáveis (cont.)

- Tipos de dados básicos

Tipo	Tamanho em bytes	Faixa
char	1	0 a 255
int	4	-2.147.483.648 a 2.147.483.647
float	4	$-3.4*10^{-38}$ a $3.4*10^{38}$ 6 dígitos de precisão
double	8	$-1.7*10^{-308}$ a $1.7*10^{308}$ 15 a 16 dígitos de precisão
void	0	Sem valor

3.1. Variáveis (cont.)

EXEMPLO: A precisão em float e double

```c
#include <stdio.h>

int main(void)
{
    float pi = 3.1415926535897932;
    double piDouble = 3.1415926535897932;

    printf("Valor de pi %.16f\n", pi);
    printf("Valor de pi mais preciso %.16lf\n", piDouble);

    return 0;
}
```

3.1. Variáveis (cont.)

EXEMPLO: Para saber o tamanho de um tipo de dado em seu computador:

```c
#include <stdio.h>

int main(void)
{
     printf("Tamanho do tipo int: %d bytes\n", sizeof(int));
     printf("Tamanho do tipo char: %d bytes\n", sizeof(char));
     printf("Tamanho do tipo float: %d bytes\n", sizeof(float));
     printf("Tamanho do tipo double: %d bytes\n", sizeof(double));

     return 0;
}
```

3.1. Variáveis (cont.)

- Modificadores de tipo

Modificador	Aplica-se a	Significa
short	int	Menor dimensão
long	int, double	Maior dimensão
signed	char, int	Com sinal
unsigned	char, int	Sem sinal

- Em C, por padrão, ao declararmos uma variável do tipo ***int***, ela será automaticamente do tipo **signed**.

- **Exemplo:**
  - long int -> 4 (bytes), faixa -2.147.483.648 a 2.147.483.647
  - unsigned int -> 2 (bytes), faixa de 0 a 65.535

3.1. Variáveis (cont.)

- Exemplo de declaração de variáveis:

```
int       valor;
float     preco;
char      genero;
double    fatorial;
long int  quantidade;
```

Cada variável possui um espaço de memória reservado, correspondente ao tamanho do seu tipo declarado.

Nome da variável

Tipo de dado

3.1. Variáveis (cont.)

- Variáveis do mesmo tipo podem ser separadas por vírgula na declaração.

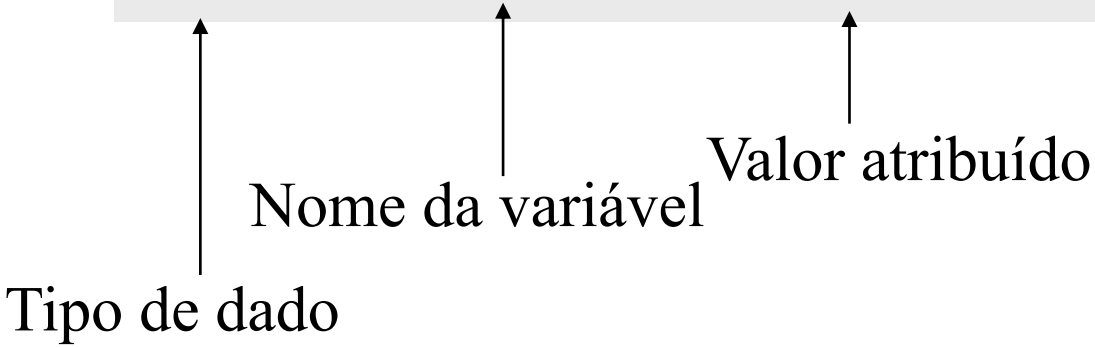
- **Exemplos:**

```
int contador, i, j, valor1=0;
double balanco_anual, balanco_mensal;
```

3.1. Variáveis (cont.)

- **As variáveis** em C **podem receber** um **valor** no momento da sua declaração. Isso é determinado pela inicialização da variável.

```
int     valor    = 0;
float   preco    = 150.50;
char    genero   = 'F';
double  fatorial = 1;
```



Se o tipo do valor atribuído à variável e o tipo da variável que recebe este valor forem distintos, pode ocorrer erro de incompatibilidade de tipo ou perda de informação.

3.1. Variáveis (cont.)

- **Origens da atribuição de conteúdo a variável:**
  - 1) nomeDaVariavel = um valor (uma constante);
  - 2) nomeDaVariavel = uma variável;
  - 3) nomeDaVariavel = uma expressão aritmética;
  - 4) nomeDaVariavel = chamada a uma função;

- ***Semântica do comando de atribuição:***
  - 1) Armazena o valor atribuído à variável a partir da posição inicial do endereço de memória reservado para ela;
  - 2) Copia o conteúdo da variável de origem armazenando-o a partir da posição inicial do endereço de memória reservado para a variável de destino;
  - 3) Após ser resolvida a expressão matemática, o resultado é armazenado a partir da posição inicial do endereço de memória reservado para a variável;
  - 4) Após a função ser executada o seu retorno é armazenado a partir da posição inicial do endereço de memória reservado para a variável.

3.1. Variáveis (cont.)

- **Modelador de tipo (*type cast/casting*)**
  - Um modelador de tipo "converte" o valor no tipo especificado.
  - Um modelador é aplicado a um valor, expressão ou variável.

Forma geral do casting:

```
(tipo de dado) valor/expressão/variável
```

Onde:

Tipo de dado *indica o tipo de dado para o qual o* ***valor, expressão ou variável*** *será transformado no tipo especificado.*

– O modelador de tipo não altera o conteúdo da variável ao qual ele é aplicado. A alteração ocorre apenas no valor e quando do seu uso.

- (int) - aplicado em um valor **float** passa a ser considerada **apenas a parte inteira do valor**.
- (float) - aplicado em um valor **int** o valor é "transformado" em **float**.

3.1. Variáveis (cont.)

- **Modelador de tipo (*type cast/casting*)**
  - Exemplo:

```c
#include <stdio.h>

int main(void)
{
    int num = 10;
    float resultado;

    resultado = (float) num / 7;
    printf("Resultado: %f\n", resultado);

    return 0;
}
```

Neste exemplo:

Sem o modelador *a Linguagem C faria uma divisão inteira entre 10 e 7. E resultado seria 1 que posteriormente seria convertido para* ***float,*** *mas continuaria a ser 1.0.*

```
Resultado: 1.000000
```

Com o modelador **num** *é transformado em* ***float*** *(10.0) e dividido por 7 (inteiro). O resultado é* ***float*** *porque um dos divisores é* ***float****.*

```
Resultado: 1.428571
```

3.2. Constantes

- Uma **constante** é um **valor armazenado** em uma área de memória **que não é alterado** durante a execução do programa.

Forma geral de declaração de constante:

```
#define identificador_da_constante valor
```

  - **#define** define que uma constante será declarada.
  - **identificador_da_constante** é o nome da constante, utilizado para acessar o seu conteúdo. O nome segue as regras para nomear identificador como de variáveis, por exemplo.
  - **valor** é o valor atribuído à constante. O conteúdo de uma constante não pode ser alterado em tempo de execução do programa. **A constante assume o tipo de acordo com o valor atribuído.** Não há **;** ao final de declaração.

3.2. Constantes (cont.)

- **Exemplos de constantes:**

```
#define PI 3.14159
#define CIDADE "Pato Branco"
#define CARACTERE 'a'
#define DIAS 7
```

3.2. Constantes (cont.)

- **Exemplo de constante de string:**

```c
#include <stdio.h>

//declaração da constante
#define CIDADE "Pato Branco"

int main(void)
{
    printf("Cidade: %s", CIDADE);

    return 0;
}
```

3.3. Função de entrada de dados

- Leitura de dados pela entrada padrão (teclado)

  `scanf ("string de controle", lista de argumentos);`

  Onde:

  - **string de controle** identifica o tipo de dado que será lido.
  - **lista de argumentos** são os identificadores das variáveis para armazenar os valores lidos.

- Exemplos:

```
scanf("%d", &idade);
scanf("%c", &caractere);
```

% *tipo de dado a ser lido*

& *é operador de endereço - indica que o valor informado será armazenado na variável*

3.3. Função de entrada de dados (cont.)

- Exemplo:

```
#include <stdio.h>

int main(void)
{
    int num;

    printf("Informe um numero: ");
    scanf("%d", &num);

    printf("valor = %d, endereco = %p", num, &num);
}
```

```
Informe um numero: 2
valor = 2, endereco = 000000000061FE1C
```

(o endereço varia conforme a memória da máquina)

3.3. Função de entrada de dados (cont.)

- Exemplo:

```
/* Mostra o uso de scanf() com várias entradas */
/* Calcula a média de quatro notas */

#include <stdio.h>

int main(void)
{
    float n1, n2, n3, n4, media;

    printf("Digite as quatro notas: ");
    scanf("%f%f%f%f",&n1,&n2,&n3,&n4);
    media = (n1 + n2 + n3 + n4)/4;
    printf("Media: %.2f\n", media);

    return 0;
}
```

3.3. Função de entrada de dados (cont.)

- Exemplo:

```c
/* Mostra a impressão formatada com o printf() */
#include <stdio.h>

int main(void)
{
    int dia, mes, ano;

    printf("Digite uma data no formato dd/mm/aaaa: ");
    scanf("%d%d%d", &dia, &mes, &ano);

    printf("A data que voce digitou foi: %02d/%02d/%04d\n", dia, mes, ano);

    return 0;
}
```

3.4. Função saída de dados

- Apresentação de dados na saída padrão (monitor de vídeo).

```
printf("texto, strings de controle e caracteres de formatação", lista de argumentos);
```

  - Onde:
    - **Texto** que será impresso literalmente ou caracteres de formatação.
    - **Strings de controle** identificam o tipo de dado contido nas variáveis cujo conteúdo será mostrado.
    - **Caracteres de formatação** indicam instruções como quebra de linha e marca de tabulação.
    - **Lista de argumentos** são os identificadores das variáveis que contêm constantes, chamadas a funções e expressões que possuem os valores a serem impressos, de acordo com os caracteres de formatação.

- Exemplo:

```
printf("Sua idade é: %d anos\n", idade);
```

3.4. Função saída de dados (cont.)

- Exemplo:

```
#include <stdio.h>

int main(void)
{
    printf ("%.2f \n", 3456.78);
    printf ("%.0f \n", 3456.78);
}
```

quantidade de caracteres depois do ponto decimal

Saída:
```
3456.78
3457
```

```
3456.78
3457
```

3.4. Função saída de dados (cont.)

- **Caracteres de formatação:**

  \n nova linha

  \t tabulação (tab)

  \b retrocesso

  \” imprimir aspas

  \\ para imprimir barra

  %% para imprimir o símbolo de %

3.4. Função saída de dados (cont.)

- **Strings de controle:**

%d ou %i	→	inteiro	exemplo
%c	→	caractere	exemplo
%f	→	float	exemplo
%lf	→	double	exemplo
%ld ou %li	→	inteiro longo	
%e	→	número ou notação científica	
%o	→	octal	exemplo
%x	→	hexadecimal	exemplo
%s	→	string (cadeia de caracteres)	

3.4. Função saída de dados (cont.)

- Exemplo de saída de **inteiros**:

```
#include <stdio.h>
int main(void)
{
    int num;

    printf("Informe um numero: ");
    scanf("%d", &num);
    printf("O valor informado é: %d", num);
}
```

  - Outros exemplos:
    - Três maneiras de exibir o número 15:

```
printf ("%5d", 15); // exibe "  15"
printf ("%05d", 15); // exibe "00015"
printf ("%-5d", 15); // exibe "15  "
```

3.4. Função saída de dados (cont.)

- Outro exemplo de saída de **inteiros**:

```
#include <stdio.h>
int main(void)
{
      int lapis=45, borrachas=2345, canetas=420, cadernos=8,
      fitas=13050;

      printf("\nLapis      %12d", lapis);
      printf("\nBorrachas  %12d", borrachas);
      printf("\nCanetas    %12d", canetas);
      printf("\nCadernos   %12d", cadernos);
      printf("\nFitas      %12d", fitas);

      return 0;
}
```

```
Lapis              45
Borrachas        2345
Canetas           420
Cadernos            8
Fitas           13050
```

3.4. Função saída de dados (cont.)

- Exemplo de saída de **caracteres**:

```
#include <stdio.h>
int main(void)
{
    char letra;
    printf("Informe um caractere: ");
    scanf("%c", &letra);
    printf("O caractere informado e: %c", letra);
}
```

  - Outros exemplos:
    - Outras maneiras de exibir o caractere “a”:

```
printf("%5c", a); // exibe "    a"
printf("%-5c", a); // exibe "a    "
```

3.4. Função saída de dados (cont.)

- Exemplo de saída de **números reais** (float):

```c
#include <stdio.h>

int main(void)
{
    float num;

    printf("Informe um numero: ");
    scanf("%f", &num);
    printf("O valor informado eh: %.2f", num);
    printf("\nO valor informado eh: %.5f", num);
    printf("\nO valor informado eh: %9.5f", num);
    printf("\nO valor informado eh: %9.1f", num);

    return 0;
}
```

```
Informe um numero: 12
O valor informado eh: 12.00
O valor informado eh: 12.00000
O valor informado eh:  12.00000
O valor informado eh:      12.0
```

3.4. Função saída de dados (cont.)

- Exemplo de saída de **números reais** double:

```
#include <stdio.h>

int main(void)
{
        double num;

        printf("Informe um numero: ");
        scanf("%lf", &num);
        printf("O valor informado eh: %.2lf", num);
        printf("\nO valor informado eh: %.5lf", num);
        printf("\nO valor informado eh: %9.5lf", num);
        printf("\nO valor informado eh: %9.1lf", num);

        return 0;
 }
```

```
Informe um numero: 8
O valor informado eh: 8.00
O valor informado eh: 8.00000
O valor informado eh:   8.00000
O valor informado eh:       8.0
```

3.4. Função saída de dados (cont.)

- Exemplo de saída em **decimal**, **hexadecimal**, **octal** e **caractere**:

```c
#include <stdio.h>

int main(void)
{
	printf("\n%d", 65);
	printf("\n%x", 65);
	printf("\n%o", 65);
	printf("\n%c", 65);

	return 0;
}
```

```
65
41
101
A
```

3.5. Operadores

- **Operadores** são elementos funcionais que atuam sobre **operandos** e produzem um determinado resultado.

- Uma expressão é composta de operadores e operandos.
  - A expressão **3 + 2** relaciona dois operandos (os números **3** e **2**) por meio do operador de adição (**+**).

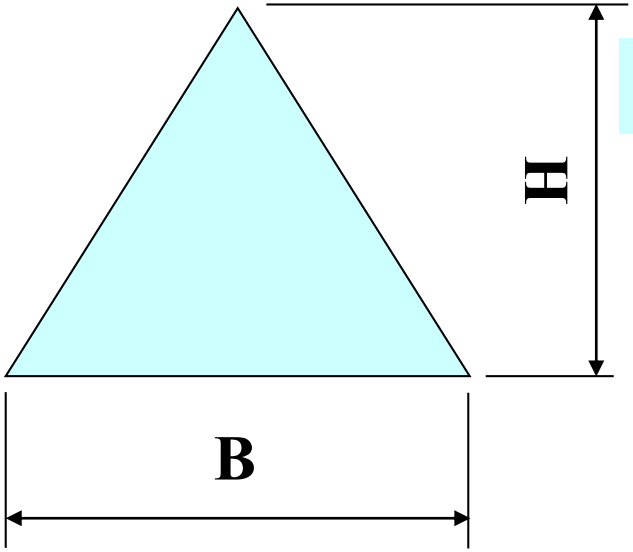
- Um operando pode ser:
  - o conteúdo de uma variável,
  - o retorno de uma função; ou
  - um valor constante.

3.5. Operadores (cont.)

- Uma **expressão** aritmética é um conjunto de variáveis e constantes numéricas relacionadas por meio de operadores aritméticos compondo uma fórmula que, uma vez avaliada, resulta um valor.

- O conceito de **expressão** aplicado à computação assume conotação mais ampla:
  - uma **expressão** é uma combinação de variáveis, retorno de funções, constantes e operadores, e que resulta em um ***valor***.
    - Variáveis, retorno de funções, constantes **são os operandos**.
    - Operações aritméticas **são os operadores**.

3.5. Operadores (cont.)



Triângulo de base (b) e altura (h)

$$\text{área} = \frac{b \times h}{2}$$

→ matemática

```
area = (b * h) / 2;
```

→ computacional

A fórmula da área do triângulo utiliza três variáveis: **b** e **h**, que contém as dimensões do triângulo, e **area** que armazena o valor calculado (o resultado da expressão).
Há, ainda, uma constante (2) e os operadores de multiplicação (*) e divisão (/).

3.5. Operadores (cont.)

- ***Tipos de Operadores***
  - ***Operadores unários*** atuam sobre um único operando. Ex.: o símbolo de subtração (-) antes de um número, cuja função é inverter seu sinal.
  - ***Operadores binários***, quando atuam sobre dois operandos. Ex.: soma, subtração, multiplicação, divisão e resto de divisão.

operadores	**resultado**
aritméticos	numérico
lógicos	lógico
relacionais	lógico

3.5. Operadores (cont.)

- **Operadores Aritméticos**

Operador	Ação
+	Adição
-	Subtração
-	Subtração (unário)
*	Multiplicação
/	Divisão
%	Resto de divisão inteira

3.5. Operadores (cont.)

- **Operadores Aritméticos – Resto da divisão**
  - O operador **%** só pode ser aplicado a valores inteiros.
  - Exemplo:

  - É o resultado da seguinte operação: 2009 % 19 = 14
  - O operador **%** é bastante usado na programação. Por exemplo:
    - **Os números pares sempre deixam resto 0 quando divididos por 2.**
    - Extrair dígitos de um valor inteiro (dividir o valor inteiro por 10 e armazenar o resto em uma variável).

3.5. Operadores (cont.)

- **Operadores Aritméticos – Resto da divisão**

```
//Exemplo do uso do operador resto de divisão (%)

#include <stdio.h>

int main(void)
{
      int num, resto;

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

      resto = num % 2;

      printf("O resto da divisao de %d eh:  %d\n", num,
resto);

     return 0;
}
```

```
Digite um numero: 5
O resto da divisao de 5 eh:  1
```

3.5. Operadores (cont.)

- **Operadores Aritméticos – Resto da divisão**

```c
//Exemplo do uso do operador resto de divisão (%) para extrair dígito

#include <stdio.h>

int main(void)
{
    int num, resto;

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

    resto = num % 10;

    printf("O resto da divisao de %d eh:  %d\n", num, resto);

    return 0;
}
```

```
Digite um numero: 123
O resto da divisao de 123 eh:  3
```

3.5. Operadores (cont.)

- **Operadores Aritméticos – Precedência**
  - Os operadores *****, **/** e **%** tem precedência sobre os operadores **+** e **–**.
    - Os parênteses podem ser usados para mudar a ordem de avaliação.
    - Exemplo: `n = (z + y) * x;`
      - A expressão entre parênteses é a primeira a ser executada.
  - A operação é sempre executada da "esquerda para a direita".
    - Exemplo: `z = x - y + 5;`
      - Na expressão, y será subtraído de x antes da execução da operação de soma.

3.5. Operadores (cont.)

- **Operadores de Incremento e Decremento**

Operador	Ação
--	Decremento
++	Incremento

3.5. Operadores (cont.)

- **Incremento e decremento**

```
x = x + 1;        é o mesmo que    x++;
x = x - 1;        é o mesmo que    x--;
```

  - Exemplos:

```
x = 10;
y = ++x;          coloca 11 em x e y também recebe 11

x = 10;
y = x++;          coloca 11 em x e y recebe 10

x = 10;
y = --x;          coloca 9 em x e y também recebe 9

x = 10;
y = x--;          coloca 9 em x e y recebe 10
```

3.5. Operadores (cont.)

- **Exemplo:**

```c
// Incremento e decremento
#include <stdio.h>

int main(void)
{
      int x = 10, y;

      y = x++;
      printf("O valor de x eh: %d\n", x);
      printf("O valor de y eh: %d\n", y);

       return 0;
}
```

```
O valor de x eh: 11
O valor de y eh: 10
```

Referências

- Introdução, história e características da linguagem C. Disponível em: https://youtu.be/sokAuwGTpnM. Acesso em: 22 fev. 2021.
- SCHILDT, H. **C Completo e total**, 3ª ed. São Paulo: Makron Books, 1996.
- MIZRAHI, V. V. **Treinamento em linguagem C**. São Paulo: Pearson Prentice Hall, 2008.
- Material elaborado com base no conteúdo disponibilizado pela professora Beatriz Borsoi.

Dúvidas

- ???

Softwares populares construídos em Linguagem C

1. Sistemas Operacionais

Unix: O sistema operacional Unix, que teve grande influência em muitos sistemas posteriores, foi originalmente desenvolvido em C. Isso ajudou a estabelecer C como a linguagem preferida para sistemas operacionais.

Linux: O kernel do Linux, que é a base de muitos sistemas operacionais modernos, incluindo distribuições como Ubuntu, Fedora e Red Hat, é escrito em C.

Windows: Embora partes do Windows sejam escritas em outras linguagens, muitas de suas bibliotecas e componentes fundamentais foram desenvolvidos em C.

2. Compiladores e Interpretes

GCC (GNU Compiler Collection): Um dos compiladores mais populares e amplamente utilizados para C, C++ e outras linguagens. O GCC é uma ferramenta fundamental no desenvolvimento de software em ambientes Unix/Linux.

LLVM: Um conjunto de ferramentas de compilação modular e reutilizável, que também inclui suporte para várias linguagens, foi desenvolvido em C++ mas utiliza partes significativas de C.

3. Software de Rede

Apache HTTP Server: Um dos servidores web mais populares, conhecido por sua robustez e flexibilidade, foi desenvolvido em C.

Postfix: Um servidor de e-mail altamente configurável e eficiente, escrito em C, que é amplamente utilizado em servidores de e-mail.

Softwares populares construídos em Linguagem C (cont.)

4. Bancos de Dados

MySQL: Um dos sistemas de gerenciamento de banco de dados mais populares, utilizado em muitas aplicações web, é parcialmente escrito em C.

SQLite: Um banco de dados leve e autônomo, amplamente utilizado em aplicações móveis e sistemas embarcados, é inteiramente escrito em C.

5. Sistemas Embarcados e Firmware

Muitas aplicações de sistemas embarcados, como controladores de dispositivos e software de automação industrial, são desenvolvidas em C devido à sua eficiência e capacidade de trabalhar diretamente com hardware.

6. Aplicações Científicas e Técnicas

MATLAB: Embora o usuário interaja com MATLAB em um nível mais alto, muitos de seus componentes e bibliotecas fundamentais são escritos em C.

Blender: Um software de modelagem 3D, animação e renderização, que utiliza C para desempenho crítico.

7. Jogos

Muitos jogos clássicos e motores de jogo (como o Quake) foram desenvolvidos em C, aproveitando a linguagem para otimização de desempenho.

8. Ferramentas de Desenvolvimento

Git: O sistema de controle de versão distribuído, amplamente utilizado em projetos de software, é escrito em C.

Make: Uma ferramenta de automação de compilação que é fundamental para o desenvolvimento de software, especialmente em projetos em C.