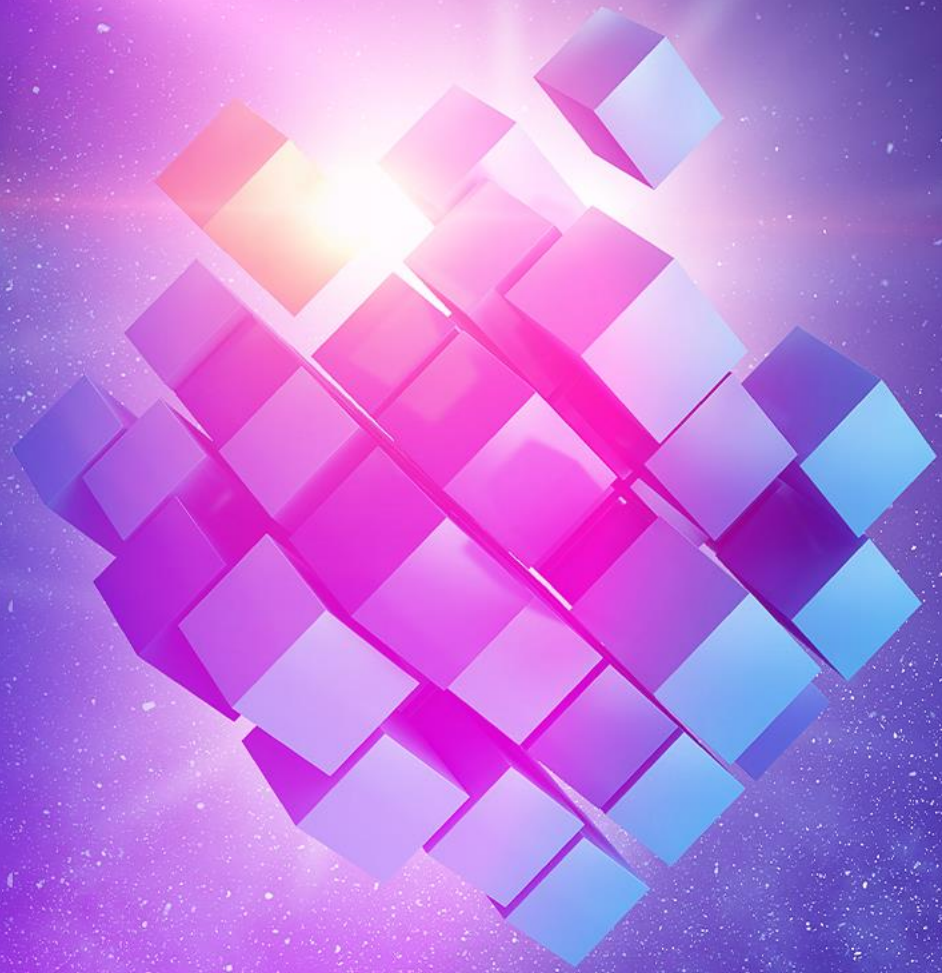
Aula Inaugural

Iremos falar sobre a questão da utilização da linguagem C



claudio.fico@animaeducacao.com.br

O QUE É A LINGUAGEM C?

C é uma linguagem de programação de computadores;
Desenvolvida em 1972 por Dennis Ritchie no Bell Lab para uso no sistema operacional Unix.

Foi amplamente aceita por oferecer aos programadores o máximo em controle e eficiência.

A linguagem C foi criada com os objetivos principais em mente:

1) Facilitar a criação de programas extensos com menos erros, recorrendo ao paradigma da programação algorítmica ou procedimental; e

2) Sobrecarregando menos o autor do compilador, cujo trabalho complica-se ao ter de realizar as características complexas da linguagem.

Softwares que usam a Linguagem C?

Sistemas Operacionais

- Windows
- Mac OS X

Navegadores

- Internet Explorer
- Mozilla Firefox

Descompactador/compactador de arquivos

- WinRAR

Editor de Imagens

- Paint
- Photoshop
- Gimp
- Corel

Jogos

- EA (NFS, FIFA, Sims)

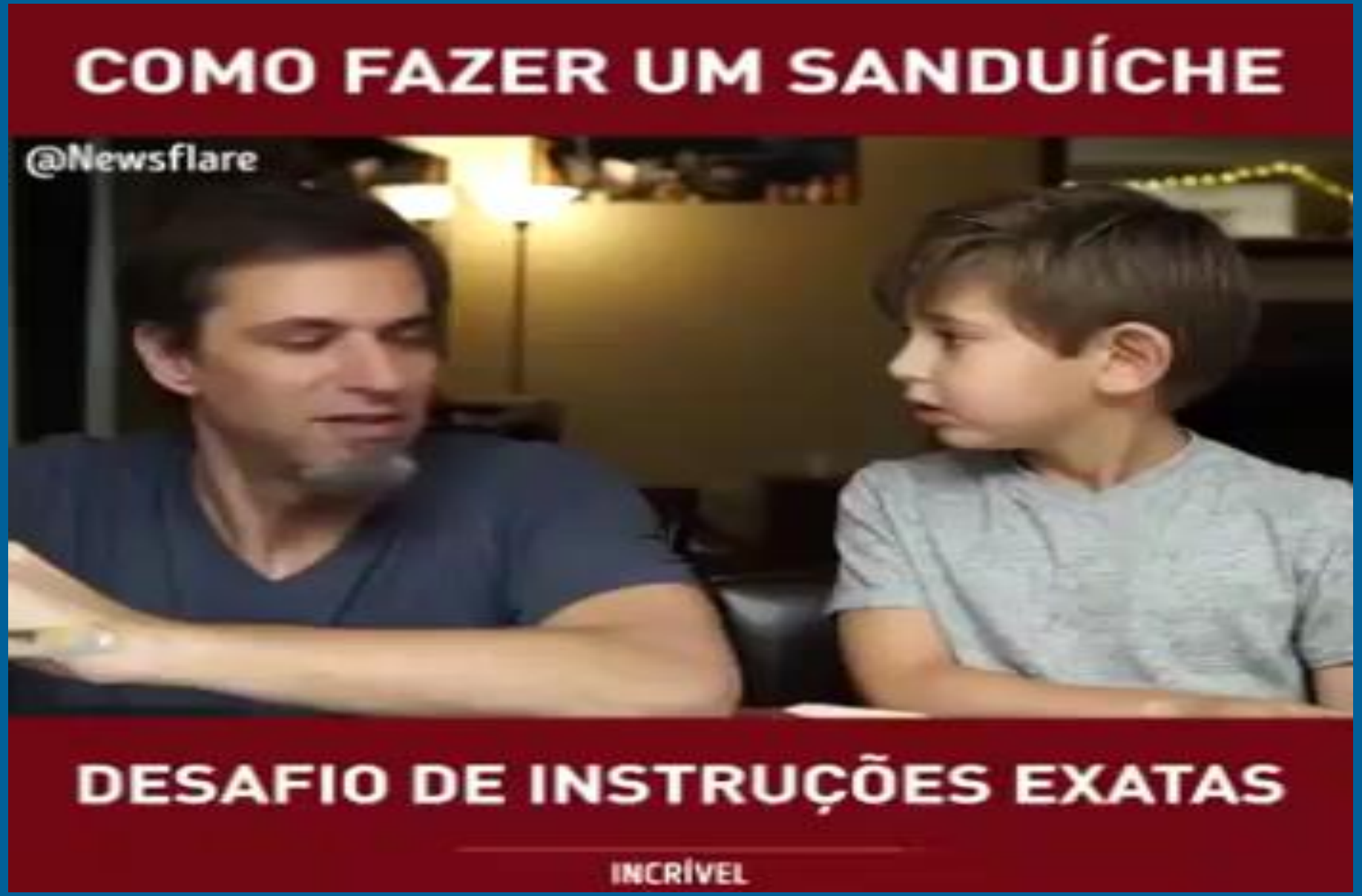
Escritório

- Office
- Adobe Acrobat

Desenvolvedor

- Visual Studio (criador de programas)

Novo Profissional da TI



As **declarações** expressam as partes do programa, podendo dar significado a um identificador, alocar memória, definir conteúdo inicial, definir funções.

As **funções** especificam as ações que um programa executa quando roda.
Uma função importante e obrigatória em todo programa em C é a função *main* (cuja tradução é *principal*). Esta será sempre a primeira função do programa a ser executada.

```
main ( )
{
    <comando>;
    <comando>;
}
```

O corpo (conteúdo) da função é delimitado por chaves { }. O código que estiver dentro das chaves será executado sequencialmente quando a função for chamada.

Estrutura de Um Programa em Linguagem C:

- Diretiva de Compilação e Biblioteca;
- Definição de Tipos de Dados;
- Protótipos de Funções;
- Funções; e
- Comentários;

Diretiva de Compilação

#include <stdio.h> → É uma diretiva de compilação. Toda diretiva de compilação inicia com **#include**. São comandos que instruem ao compilador a realização de tarefas antes de realizar a compilação.

Biblioteca

No arquivo **stdio.h** existem declarações de funções úteis para entrada e saída de dados, onde:
std = standard, que significa padrão, em inglês;
io = Input/Output, entrada e saída.
.h = header (cabeçalho)

Existem muitas outras bibliotecas que serão vistas ao longo da disciplina de Algoritmos.

Variáveis e Constantes:

Variáveis e constantes são os elementos básicos que um programa manipula.

Variáveis → São pequenos pedaços alocados na memória do computador para guardar informações que mudam no decorrer do programa.

Constantes → São pequenos pedaços alocados na memória do computador para guardar informações que **NÃO** mudam no decorrer do programa.

Tipos de Dados

Os dados podem assumir cinco tipos básicos em C que são:

char: Caractere: O valor armazenado é um caractere.

int: Número inteiro é o tipo padrão e o tamanho do conjunto que pode ser representado normalmente depende da máquina em que o programa está rodando.

float: Número em ponto flutuante de precisão simples. São conhecidos normalmente como números reais.

double: Número em ponto flutuante de precisão dupla

void: Este tipo serve para indicar que um resultado não tem um tipo definido. Uma das aplicações deste tipo em C é criar um tipo vazio que pode posteriormente ser modificado para um dos tipos anteriores.

Nomes das Variáveis

Existem algumas regras básicas que regulam a o batismo de variáveis.
Estas regras básicas são:
Todo nome só pode conter letras e dígitos;
O caractere "_" é contado como uma letra;
Todo primeiro caractere deve ser sempre uma letra;
Letras maiúsculas e minúsculas são consideradas caracteres diferentes;
Palavras reservadas não podem ser usadas como nome de variáveis.

Palavras reservadas:

auto / break / case / char / const / continue / default / do / double / else / extern / for / float / short / struct / union / unsigned / void / while

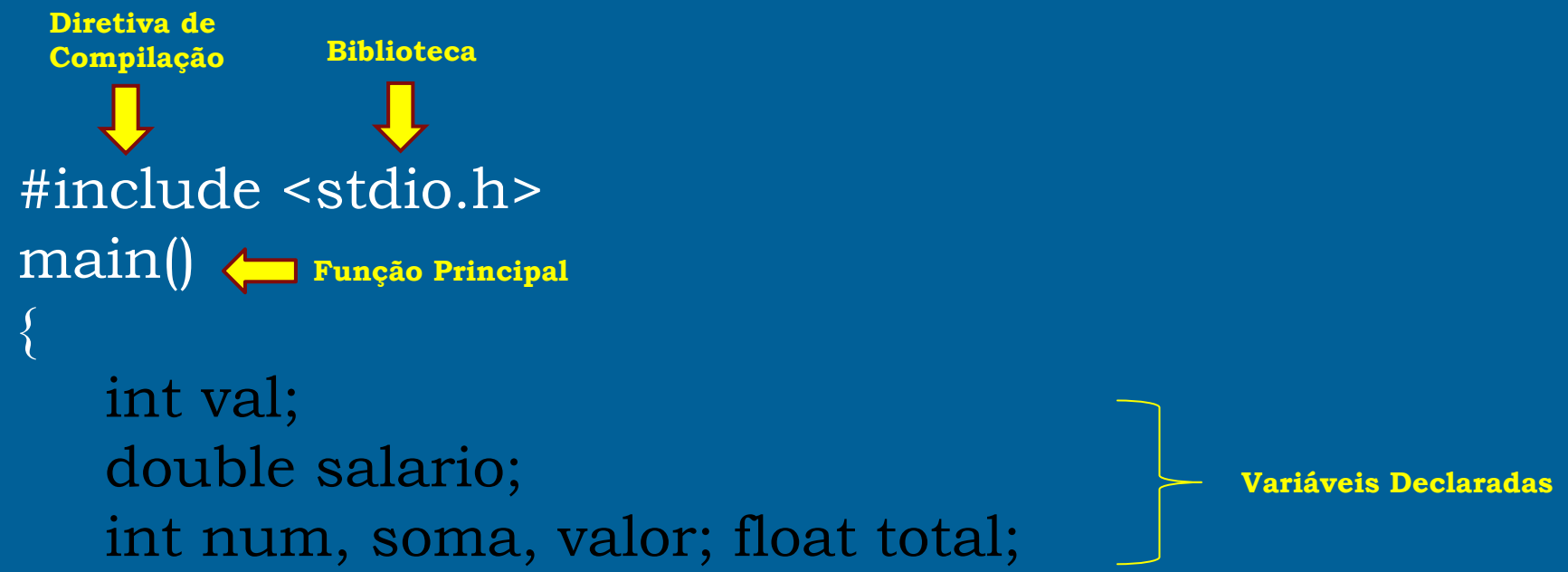
É boa a política de escolher nomes que significam alguma coisa e indiquem a função da variável.

Exemplo:

valor, soma, total, nome, raio.

A Linguagem C é "case sensitive"

Um ponto de suma importância: C é *"case sensitive"*, ou seja, maiúsculas e minúsculas fazem diferença.
Se declararmos uma variável com o nome "soma" ela será diferente de "Soma", "SOMA", "SoMa" ou "sOmA".
Também se aplica para comandos e funções.



Variáveis Locais e Globais

Variável Local → é aquela declarada dentro do corpo de uma certa função e somente pode ser utilizada por aquela função e nenhuma outra mais.

Variável Global → poderá ser utilizada por todas as funções existentes em seu programa, é declarada fora, antes do início do corpo da função principal do programa main().

Exemplos de varíavel global

```
#include <stdio.h>
int a; // variável declarada como GLOBAL (está antes
    da função main())
main ()
{
a = 3;
}
```

Exemplos de varíavel local

```
#include <stdio.h>
main ()
{
int a; // variável declarada como LOCAL (está após a
função main())
a = 3;
}
```

Exemplo de Declaração de Variáveis Constantes

```
#include <stdio.h>

#define icms 0.18; //declaração da constante

main()
{
 float preco_produto, valor_icms; // declaração
                    variável local
 preco_produto = 50;
 valor_icms = preco_produto * icms;
 printf("Valor de imposto a ser pago: R$ %f", valor_icms);
}
```

Exemplo de Atribuição de Valores às Variáveis

Após ser declarada, a variável pode receber valores. O operador de atribuição "=" indica que o valor à direita será atribuído à variável.

Durante a declaração da variável

```
#include <stdio.h>
main()
{
    int val = 0, num = 10; //declaração do valor na definição da variável
    float raio = 2.54;
    char sexo = 'm';
}
```

```
#include <stdio.h>
#define icms 0.18; //declaração da constante
main()
{
   float preco_produto, valor_icms; // declaração variável local
    preco_produto = 50; // atribuição com número fixo
    icms_i = icms; // atribuição com outra variável
    valor_icms = preco_produto * icms; // atribuição com fórmula
 }
```

Operadores

Operadores Relacionais	
=	Atribuição
!=	Diferente
>	Maior
<	Menor
<=	Menor igual
>=	Maior igual
==	Igual
%	Resto da divisão entre números inteiros

Operadores Lógicos	
&&	And (e)
||	Or (ou)
!	Not (não)

Operadores Aritméticos

Aritmético	Operação	Prioridade
+	Adição	5
-	Subtração	5
%	Resto da divisão	4
*	Multiplicação	3
/	Divisão	3
++	Incremento	2
--	Decremento	2
+	Manutenção do sinal	1
-	Inversão do sinal	1

Operadores Aritméticos

Operadores e Expressões

Expressões: As expressões combinam operandos e operadores para produzir um único resultado. Ex: a + b * c – d

Operadores de Atribuição (=) – O valor da expressão da direita é atribuído à variável da esquerda. Ex: soma = a + b; a = b = c = d = 10;

Operadores Aritméticos (* , / , % , + , -)

Ex: a = 3 + 2 * (b = 7/2); → b = 7 / 2 e a = 3 + 2 * b

printf (“%d”, -3 + 4 * 5 – 6) → 11

Aritméticos de Atribuição: (+= , -= , *= , /=)

Ex: x = x + 5; => x += 5;

x = x – 5; => x -= 5;

x = x * 5; => x *= 5;

x = x / 5; => x /= 5;

PROGRAMANDO EM C

Funções de Saída

Função printf()

Através da função pré-definida printf(), cujo protótipo está contido também no arquivo *stdio.h*, poderemos imprimir na tela informações.

Sintaxe:

printf (“Expressão”);
ou
printf (“Expressão” , lista de argumentos);

Exemplo:

printf (“Volta ao Mundo!”);

Expressão → Mensagens que serão exibidas.
Lista de Argumentos → Pode conter identificadores de variáveis, expressões aritméticas ou lógicas e valores constantes.

Acentuação

Você vai utilizar a IDE DEV C++ para criar um código para o terminal no Windows e, ao dar o comando prinft com um texto acentuado aparecem caracteres estranhos no lugar dos acentos.
Para resolver este problema basta você usar o comando de regionalização do C para que não somente acentue as palavras corretamente, mas que mostre datas e horas em português, por exemplo:

```
#include <stdio.h>
#include <locale.h>
main()
{
    setlocale (LC_ALL, "Portuguese");
    printf("Alô mundo!");
    printf ("E aí, você está bem? ");
    printf ("Boa Apresentação!");
}
```

Impressão de Tipos de Dados e caracteres Especiais

Código	Tipo	Elemento Armazenado
%c	char	Um Único Caractere
%s	char	Uma Cadeia de Caracteres
%d ou %i	int	Um Inteiro
%f	flout	Um Número em Ponto Flutuante
%lf	double	Ponto Flutuante com Dupla Precisão

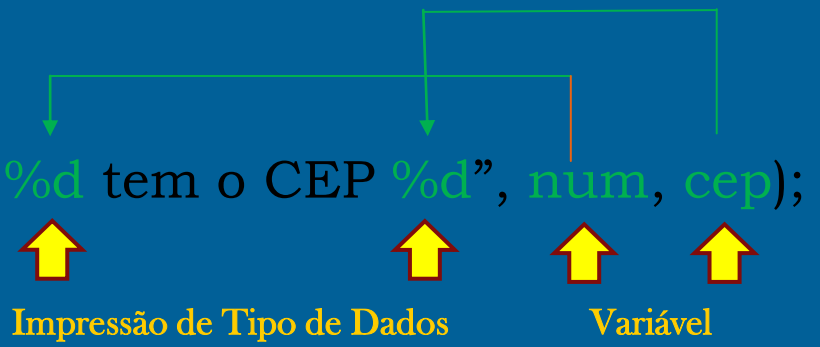
Para cada impressão de tipo de dados deve haver uma variável na função printf().

Código	Ação
\n	Leva o cursor para a próxima linha
\t	Executa uma tabulação
\b	Executa um retrocesso
\f	Leva o cursor para a próxima página
\a	Emite um sinal sonoro (beep)
\”	Exibe o caractere "
\\	Exibe o caractere \
%%	Exibe o caractere %

EXEMPLOS:

```
#include <stdio.h>
main()
{
int dias;
dias = 30;
printf (“Volta ao Mundo em %d dias”, dias);
}
```


```
#include <stdio.h>
main()
{
int num, cep;
num = 28;
Cep = 22786021;
printf (“A casa de numero %d tem o CEP %d”, num, cep);
}
```



Funções de Entrada de Dados

Função scanf()

Ela é o complemento de printf() e nos permite ler dados formatados da entrada padrão (teclado).
A lista de argumentos deve consistir nos endereços das variáveis. A linguagem C oferece um “operador de endereço” e referenciado pelo símbolo "&".

Operador de endereço &:
Antes de cada variável para indicar o endereço de memória no qual o conteúdo da variável estará armazenado. Este é uso obrigatório quando tratamos variáveis do tipo numérica (inteira ou real).

Sintaxe:
scanf (“impressão de tipo de dado”, &variavel);

Exemplo:
int val;
scanf (“%d”, &val);
%d → indicativo do tipo, neste caso do tipo inteiro.
&→ operador utilizado para obter o endereço de memória da variável.
val→ variável que receberá a informação via teclado.

EXEMPLO:

```
#include <stdio.h>
#include <stdlib.h>   //biblioteca para o uso do comando system
main()
{
      int valor1 = 0, valor2 = 0, resultado = 0;
      system ("cls"); //função para limpar a tela

printf ("Digite o 1º valor:");
      scanf ("%d", &valor1);

      printf ("Digite o 2º valor:");
      scanf ("%d", &valor2);

      resultado = valor1 + valor2;

      printf ("\nSoma dos valores: %d", resultado);
}
```

// → é usado para fazer comentário na linha do programa

EXERCÍCIOS:

1 – Fazer um programa para que o usuário possa digitar dois valores inteiros e ao final o programa deverá exibir o resultado da subtração desses valores.

2 - Fazer um programa para que o usuário possa digitar três valores tipo float e ao final o programa deverá exibir o resultado da multiplicação desses valores.

3 - Fazer um programa para que o usuário possa digitar quatro valores tipo float e ao final o programa deverá exibir o resultado da media dos dois primeiros valores e o resultado da media dos dois últimos valores.

Conversão de Tipo de Dados

Muitas vezes temos a necessidade de converter um tipo de dado. Para isso é preciso converter o valor que está na variável.
A conversão é feita colocando o tipo desejado antes do valor queremos converter.

Exemplo:

```
#include <stdio.h>
main()
{
	float var1 = 0;
	int var2 = 10;
	var1 = float (var2); // realiza a conversão do valor inteiro para float
	printf (“\nO Valor com decimal eh: %f", var1);

	int var3 = 0;
	float var4 = 0;
	var4 = 15.33;
	var3 = int (var4); // realiza a conversão do valor float para inteiro
	printf ("\nO Valor Inteiro eh: %d", var3);
}
```

Comando Condicional If

O comando if é uma estrutura de decisão que permite ou não que uma sequência de comandos seja executada (ou não executada), dependendo do resultado de uma condição pré-estabelecida.

Sintaxe:

```
if (<condição>) // uso do parênteses é obrigatório
    <único comando>;
```

ou

```
if (<condição>)
{
    <comando 1>;
    <comando 2>;
    <comando 3>;
}
```

A expressão sempre será avaliada logicamente (verdadeiro ou falso).

Relembrando os Operadores para uso do Comando IF

Operadores Relacionais	
!=	Diferente
<	Menor
>	Maior
<=	Menor igual
>=	Maior igual
==	Igual
%	Resto da divisão entre números inteiros

Operadores Lógicos	
&&	and (e)
||	or (ou)
!	not (não)

Comando Condicional If Else

O comando if pode decidir entre duas sequências de comandos qual vai ser a executada.

Sintaxe:

```
if (<condição>) // caso a expressão verificada retorne verdadeiro
{
      <comando>;
      <comando>;
      <comando>;
}
else // caso a expressão verificada retorne falso
{
      <comando>;
      <comando>;
}
```

Sempre que houver mais de uma linha ligada ao IF ou ao ELSE é fundamental o uso das chaves para determinar que o bloco de comandos pertence àquela situação de verdadeiro e/ou falso;

EXEMPLO:

Definir qual é o menor número digitado pelo usuário:

```
#include <stdio.h>
main()
{
    int val1, val2, menor;
    printf (“Digite o primeiro número”);
    scanf (“%d”, &val1);
    printf (“Digite o segundo número”);
    scanf (“%d”, &val2);

    if (val1 < val2)
       printf (“O menor número digitado foi %d”, val1);
    else
       printf (“O menor número digitado foi %d”, val2);
}
```

Comando Condicional If Else if

O comando if pode decidir entre duas sequências de comandos qual vai ser a executada.

Sintaxe:

```
if (<condição>) // caso a expressão verificada retorne verdadeiro
{
      <comando>;
      <comando>;
}
else if (<condição>)// caso a expressão verificada retorne falso, também poderá fazer uma nova condição com o uso do else if.
{
      <comando>;
      <comando>;
}
```

Sempre que houver mais de uma linha ligada ao IF ou ao ELSE é fundamental o uso das chaves para determinar que o bloco de comandos pertence àquela situação de verdadeiro e/ou falso;

EXEMPLO:

Verificar se o número digitado é par.

```
#include <stdio.h>
main()
{
    int val;
    printf ("Digite um número inteiro: ");
    scanf ("%d", &val);

    if (val %2 == 0)
        printf ("%d e par \n", val);
    else
        printf ("%d e impar \n", val);
}
```

Não há nada que impeça que mesmo com uma única linha de comando atrelada ao IF e/ou ELSE seja empregado o uso das chaves.

EXEMPLO:

```
#include <stdio.h>
main()
{
      int idade = 0;
      printf ("Informe sua idade: ");
      scanf ("%d", &idade);
      if (idade < 20)
      {
            if (idade < 13)
                  printf ("Infantil.");
            else
                  printf ("Adolescente");
       }
      else if (idade < 50)
            printf ("Adulto");
      else
            printf ("Idoso");
}
```

EXEMPLO COM OPERADOR LÓGICO (&& - E):

```
#include <stdio.h>
main()
{
    int idade = 0;
    char sexo;
    printf ("Informe seu sexo: ");
    scanf ("%s", &sexo);

    printf ("Informe sua idade: ");
    scanf ("%d", &idade);

    if ((idade < 20) && (sexo == 'm'))
        printf ("Ação Permitida");
    else
        printf ("Ação Não Permitida");
}
```

EXEMPLO COM OPERADOR LÓGICO (|| - OU):

```
#include <stdio.h>
main()
{
    int idade = 0;
    char sexo;
    printf ("Informe seu sexo: ");
    scanf ("%s", &sexo);

    printf ("Informe sua idade: ");
    scanf ("%d", &idade);

    if ((idade >= 20) || (sexo == 'm'))
        printf ("Ação Permitida");
    else
        printf ("Ação Não Permitida");
 }
```

Exemplo com Operadores Lógicos (&& - E / || - OU):


```
#include <stdio.h>
main()
{
    int idade = 0;
    char sexo;
    printf ("Informe seu sexo: ");
    scanf ("%s", &sexo);

    printf ("Informe sua idade: ");
    scanf ("%d", &idade);

    if (((idade >= 20) && (idade <= 50)) || (sexo == 'm'))
        printf ("Ação Permitida");
    else
        printf ("Ação Não Permitida");
 }
```

Exemplo com Operadores Lógicos (&& - E / || - OU):

```
#include <stdio.h>
main()
{
    int idade = 0;
    char sexo;
    printf ("Informe seu sexo: ");
    scanf ("%s", &sexo);

    printf ("Informe sua idade: ");
    scanf ("%d", &idade);

    if (idade >= sexo) // Erro! Comparação equivocada
entre as variáveis. Tipos diferentes.
        printf ("Ação Permitida");
    else
        printf ("Ação Não Permitida");
 }
```

```
#include <stdio.h>
main()
{
      int idade1 = 0, idade2 = 0, idade3 = 0;
      char sexo;
      printf ("Informe uma idade: ");
      scanf ("%d", &idade1);

      printf ("Informe uma idade: ");
      scanf ("%d", &idade2);

      printf ("Informe uma idade: ");
      scanf ("%d", &idade3);

      if (idade1 >= idade2 >= idade3) // Isso não funciona!
            printf (“Idade 3 é a menor idade");
      else
            printf (“Idade 3 não é a menor idade");
 }
```

```
#include <stdio.h>
main()
{
      int idade1 = 0, idade2 = 0, idade3 = 0;
      char sexo;
      printf ("Informe uma idade: ");
      scanf ("%d", &idade1);

      printf ("Informe uma idade: ");
      scanf ("%d", &idade2);

      printf ("Informe uma idade: ");
      scanf ("%d", &idade3);

      if ((idade1 >= idade2) && (idade2 >= idade3)) // Isso funciona!
            printf (“Idade 3 é a menor idade");
      else
            printf (“Idade 3 não é a menor idade");
 }
```

Comando Goto

Realiza o desvio condicional da execução do programa para um outro ponto determinado no programa através do **rótulo ou label**.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int idade = 0;

    inicio: // o rótulo precisa ser seguido dos “dois pontos”
    printf ("Informe sua idade: ");
    scanf ("%d", &idade);
    if (idade <= 0)
    {
        printf (“Erro!, Digite Novamente!");
        system(“cls”);
        goto inicio; // volta a execução para o rótulo inicio
    }
}utilização de chaves.
```

Switch Case

É muito utilizado, principalmente para uso em estruturas de menu.
O conteúdo de uma variável é comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado.
A instrução break termina a execução do switch e o programa continua a executar na instrução seguinte.
O comando default exibe uma mensagem, caso nenhuma das alternativas anteriores seja verdadeira.
Não são aceitas expressões condicionais no comando switch...case, somente são aceitos valores constantes. A expressão deve ter um valor primitivo básico, ou seja, char ou int com suas variações, não podendo portanto ser strings, float, double ou ponteiros.

Pode-se trabalhar com outros comandos entre o **case e o break.**

```
switch (variável)
{
      case constante1:
            <instruções>;
            break;
      case constante2:
            <instruções>;
            break;
      default
            <instruções>;
}
```

Exemplo:

```
#include <stdio.h>
main()
{
  int valor;
   printf ("Digite um valor de 1 a 3: ");
  scanf ("%d", &valor);

  switch ( valor )
  {
    case 1 :
       printf ("Domingo\n");
       break; // usado para interromper a ação e não verificar os demais
    case 2 :
       printf ("Segunda\n");
       break;
    case 3 :
       printf ("Terça\n");
       break;
    default :
       printf ("Valor invalido!\n");
  }
}
```

Exemplo:

```
#include <stdio.h>
main()
{
  char oper;
  float res = 0;
  printf ("Digite um sinal: ");
  scanf ("%s", &oper);

switch (oper)
{
      case '+' :
            res = 5 + 1; printf ("valor de res eh: %f", res); break;
      case '-' :
             res = 5 - 1; printf ("valor de res eh: %f", res); break;
      case '*':
            res = 5 * 1; printf ("valor de res eh: %f", res); break;
      case '/':
             res = 5 / 1; printf ("valor de res eh: %f", res); break;
      default:
            printf ("Operador desconhecido !");
}
}
```

Exercícios:

1) Fazer um programa para calcular o prêmio proporcional de um ganhador do concurso, sendo que o total do prêmio é de cem mil, o primeiro ganhador recebe 46%, o segundo 32% e o terceiro o restante. O programa só poderá aceitar opções para os 3 ganhadores, caso contrário exibir a mensagem de erro.

2) Fazer um programa de calculadora que executará seus cálculos com base em 6 operações: soma; subtração; multiplicação; divisão; potenciação; raiz quadrada. Caso seja digitado um valor diferente de seis, exibir mensagem de erro.

Raiz Quadrada: sqrt(variavel);
Potenciação: pow(base, exp);

FUNÇÕES

Funções

Conjunto de comandos agrupados em um bloco que recebe um nome e através deste pode ser ativado.

Por que usar Funções?

- Para permitir o reaproveitamento de código já construído;
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
- Para permitir a alteração de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas **dentro** da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
- Para facilitar a leitura do programa-fonte de uma forma mais fácil;
- Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.

Parâmetros

Estes parâmetros possibilitam que se definida sobre quais dados a função deve operar. Para definir os parâmetros de uma função o programador deve explicitá-los como se estive declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas.

Exemplo:
A seguir temos a **função SOMA** que possui dois parâmetros, sendo o primeiro um **int** e o segundo outro **int**.

```
int SOMA (int val1, int val2)  // basta separar por vírgulas
{
   int resultado;      // declaração de variáveis
   resultado = val1 + val2;
   printf ("A soma de %d com %d é %.d", val1, val2, resultado);
}
```

Os parâmetros da função na sua declaração são chamados **parâmetros formais**. Na chamada da função os parâmetros são chamados **parâmetros atuais**.
Os parâmetros são passados para uma função de acordo com a sua **posição**. Ou seja, o primeiro parâmetro atual (da chamada) define o valor o primeiro parâmetro formal e assim por diante. Os nomes dos parâmetros na chamada não tem relação com os nomes dos parâmetros na definição da função.

Exemplo:

```
#include <stdio.h>
int soma (int num_1, int num_2)  // basta separar os parâmetros por
virgulas
{
   int resultado;      // declaração de variáveis
   resultado = num_1 + num_2;
   printf ("A soma de %d com %d é %d", num_1, num_2, resultado);
}

main()
{
    int val_1;
    int val_2;
    val_1 = 10;
    val_2 = 12;
    soma (val_1, val_2);
}
```

Exemplo:

```
#include <stdio.h>
int soma (int num_1, int num_2)
{
   int valor;
   valor = num_1 + num_2;
   return valor;
}

main()
{
    int val_1;
    int val_2;
    int resultado;
    printf (“Digite o primeiro valor: “);
    scanf( “%d”, &val_1);
    printf (“Digite o segundo valor: “);
    scanf( “%d”, &val_2);

    resultado = soma (val_1, val_2);  // Chamada da função SOMA()
    printf ("A soma de %d com %d é %d", val_1, val_2, resultado);
}
```

Exercício:

1) Fazer um programa para que o usuário possa digitar duas notas. O programa tem a finalidade de realizar o cálculo da média aritmética.
   Formula: media = ((nota_1 + nota_2) /2)

Observações:

* As notas não poderão ter valores negativos, será preciso fazer a verificação de cada entrada de dados. Havendo erro, dar mensagem de erro e retornar para nova digitação daquela nota.

* O cálculo da média deverá ser realizado em uma nova função. Ainda na função será preciso exibir uma mensagem com o valor da media.

```
#include <stdio.h>
float media (float n1, float n2)
{
	float result;
	result = ((n1 + n2) / 2);
	printf("a media entre %f e %f eh %f", n1, n2, result);
}

main()
{
float n1, n2;
nota1:
printf("digite a primeira nota: ");
scanf("%f", &n1);
if (n1 < 0)
{	printf ("Erro! Informe uma nota valida\n");
	goto nota1;
}
nota2:
printf("digite a segunda nota: ");
scanf("%f", &n2);
if ((n2 < 0))
{	printf ("Erro! Informe uma nota valida\n");
	goto nota2;
}
media (n1, n2);
}
```

Exercício:

2) Fazer um programa para que o usuário possa utilizar uma calculadora com as funções de: soma; subtração; multiplicação; divisão;
O usuário deverá digitar sempre 2 números para poder realizar as operações aritméticas.

Observações:

* Os valores digitados não poderão ter valores negativos e será preciso fazer a verificação de cada entrada de dados. Havendo erro, dar mensagem de erro e retornar para nova digitação daquele valor.

* Será preciso criar uma nova função para calcular as operações aritméticas. A função também deverá exibir o resultado final da operação aritmética.

* Na função main() será preciso que haja uma estrutura para que o usuário possa escolher qual operação aritmética a função irá realizar.

Funções Matemáticas

Necessário o uso da biblioteca **math.h**

Potências

***pow ()*:** Retorna o valor da base elevada ao expoente. Recebe dois argumentos do tipo double, o primeiro é a base e o segundo o expoente. Por exemplo: se quisermos saber o resultado da operação 2^{10}, faríamos **pow (2, 10)**.

Raiz Quadrada

***sqrt ()*:** Retorna o valor da raiz quadrada. Recebe como argumento um double do qual ele deve extrair a raiz.

EXEMPLO:

```
#include <stdio.h>
#include <math.h>
main()
{
      double base = 0, exp = 0;
      printf ("Informe a base: ");
      scanf ("%lf", &base);
      printf ("Informe o Expoente: ");
      scanf ("%lf", &exp);
      printf (“O resultado e: %lf“, pow(base, exp));
}
```

```
#include <stdio.h>
#include <math.h>
main()
{
      double num = 0;
      printf ("Informe o numero: ");
      scanf ("%lf", &num);
      printf (“O resultado e: %lf“, sqrt(num));
}
```

Função strcmp

A função utilizada para comparar o conteúdo de uma string é a **strcmp()** que está definida na biblioteca **string.h**.

Obs: **strcmp()** compara o **conteúdo**, ou seja, se as palavras são iguais, e não se possuem o mesmo tamanho.

Sintaxe:
strcmp (variavel_1, variavel_2); ou
strcmp (variavel, “informação”)

Onde variavel_1 e variavel_2 são variáveis do tipo char que devem ser comparadas.

A função **strcmp()** pode retornar um valor zero, um ou menos um.
Quando as palavras comparadas são iguais, a função retorna 0.
Quando as palavras comparadas são diferentes, a função retorna um ou menos um.

EXEMPLO:

```
#include <stdio.h>
#include <string.h>
main()
{
char palavra[7];
printf (“Digite a palavra: ”);
scanf (“%s”, &palavra);

if (strcmp (palavra, “pequeno”) == 0)
    printf(“informação correta!”);
else
    printf(“informação incompatível!”);
}
```

EXEMPLO:

```
#include <stdio.h>
#include <string.h>
main()
{
char palavra_I[7], palavra_II[7];
printf (“Digite a primeira palavra: ”);
scanf (“%s”, &palavra_I);
printf (“Digite a segunda palavra: ”);
scanf (“%s”, &palavra_II);

if (strcmp (palavra_I, palavra_II) == 0)
    printf(“informação correta!”);
else
    printf(“informação incompatível!”);
}
```

Funções de Entrada de Dados

Função gets()

Permite a entrada de dados via teclado para caso o usuário tenha a necessidade de escrever uma frase e utilizar espaço entre as palavras.

Sintaxe:

gets (variavel);

Exemplo:

```
#include <stdio.h>
main ()
{
    char nome[30];
    gets (nome);
}
```

gets() não funciona para variável do tipo CHAR que não tenha quantidade de caractere.

FUNÇÃO STRLEN

A função **strlen()** retorna o comprimento da string fornecida. O terminador nulo não é contado. Isto quer dizer que, de fato, o comprimento do vetor da string deve ser um a mais que o inteiro retornado por **strlen()**.

Uso obrigatório da biblioteca <string.h>

Sintaxe:

strlen (variavel);

Exemplo:

```
#include <stdio.h>
#include <string.h>
main ()
{
      int tam;
      char nome[50];
      printf ("Entre com um nome: ");
      gets (nome);
      tam = strlen (nome);
      printf ("\nO nome digitado tem tamanho %d", tam);
}
```

FUNÇÃO STRCPY

Esta função é utilizada para inserir uma informação para uma variável do tipo CHAR.

Uso obrigatório da biblioteca <string.h>

Sintxe:

strcpy (str1, “informação”);

Exemplo:

```
#include <stdio.h>
#include <string.h>
main()
{
    char str1[40], str2[10];
    strcpy (str1, "TESTE"); // copia para a variável
                str1 o conteúdo TESTE
    printf ("A palavra eh: %s", str1);
}
```

Exemplo:

```
#include <stdio.h>
#include <string.h>
main()
{
char str1[10], str2[10];
printf ("Digite uma informação: ");
scanf ("%s", &str2);
strcpy (str1, str2); // atribui o conteudo de str2 para a variável str1
printf ("A palavra eh: %s", str1);
}
```

Função strcat

Esta função é utilizada para concatenar (unir / juntar) duas strings. A segunda string será adicionada no final da primeira string indicada. Lembre-se que a soma dos valores de caracteres da str1 + str2 não podem exceder o tamanho da str1. Podemos também substituir str2 por um conjunto de caracteres manualmente.

Uso obrigatório da biblioteca <string.h>

Sintxe:

```
strcat (str1,str2);
```

Exemplo:

```
#include <stdio.h>
#include <string.h>
main()
{
char str1[40], str2[10];
strcpy (str1, "TESTE"); // atribui a palavra TESTE para a variável str1
strcpy (str2, " INICIAL"); // atribui a palavra INICIAL para a variável str2
strcat (str1, str2); // concatena o conteúdo de str1 com str2
printf ("%s", str1);
}
```

Resultado na tela: TESTE INICIAL

Exemplo:

```
#include <stdio.h>
#include <string.h>
main()
{
    char str1[40], str2[10]
    gets (str1);
    gets (str2);
    strcat (str1, str2);

    printf ("A frase eh: %s", str1); // str1 recebe o
conteúdo da variável str2.
}
```

Exemplo:

```
#include <stdio.h>
#include <string.h>
main()
{
    char str1[40], str2[10], str3[10];
    strcpy (str1, "TESTE");
    strcpy (str2, " INICIAL");
    strcpy (str3, " FEITO");
    strcat (str1, str2);
    strcat (str1, str3);
    printf ("A frase eh: %s", str1); // str1 recebe o
conteúdo das variáveis str2 e str3.
}
```

Resultado na tela: TESTE INICIAL FEITO

ESTRUTURA DE REPETIÇÃO

Comando While

Uma maneira possível de executar um laço é utilizando o comando while. Ele permite que o código fique sendo executado numa mesma parte do programa de acordo com uma determinada condição. Ele é executado desde que a condição seja verdadeira.
Testa a condição antes de executar o laço.

Sintaxe:

```
while (condição)
{
      <comando>;
      <comando>;
}
```

Exemplo:

```
main()
{
int i;
i = 0;
while (i<=5) // (i <=5) é a condição. Quantas vezes a estrutura irá repetir
      {
         printf("Número %d\n", i);
         i++; // contador. A cada passagem o i é incrementado com mais 1
       }
}
```

Exemplo:

```
#include<stdio.h>
main()
{
int i;
i = 3;
while (i > 0) // (i > 0) é a condição. Quantas vezes a estrutura irá repetir
      {
         printf("Número %d\n", i);
         i--; // contador. A cada passagem o i é decrementado com menos 1
      }
}
```

Exemplo:

```
#include <stdio.h>
main()
{
int i, x;
i = 0;
x = 1;

while ((i<=5) || (x < 3))
{
        printf("Valor de i eh:  %d\n", i);
        printf("Valor de x eh:  %d\n", x);
        i++;
        x++;
}

}
```

VETOR (ARRAYS)

Em diversas situações os tipos básicos de dados (int, float, char, ...) não são suficientes para representar a informação que se deseja armazenar.

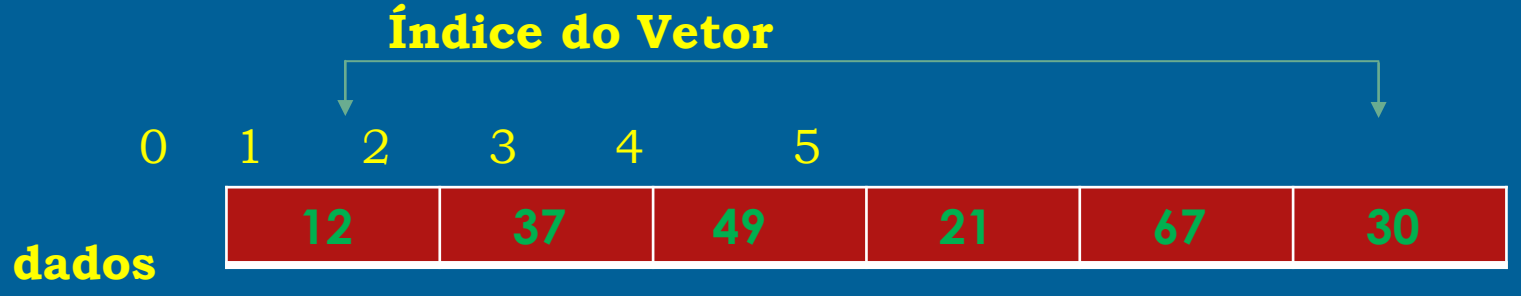
Existe a possibilidade de construção de novos tipos de dados a partir da composição (ou abstração) de tipos de dados primitivos. Esses novos tipos têm um formato denominado ESTRUTURA DE DADOS, que define como os tipos primitivos estão organizados.

Sintaxe:

tipo_primitivo identificador[qtde_elementos];

Exemplo:

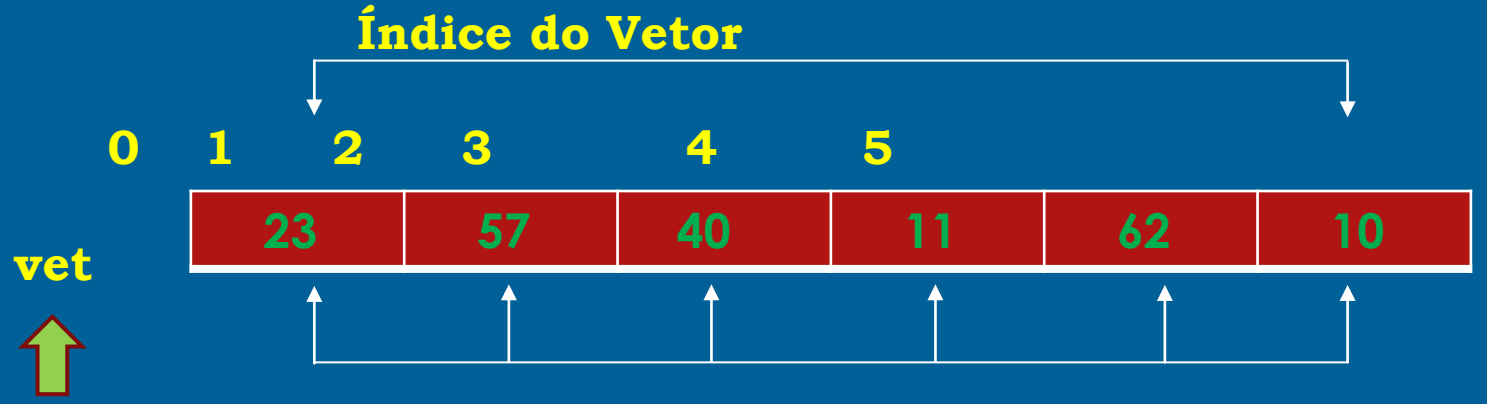
int dados[6]; // vetor com 6 elementos do tipo int. Os índices vão de 0 a 5.
dados → Nome do Vetor;



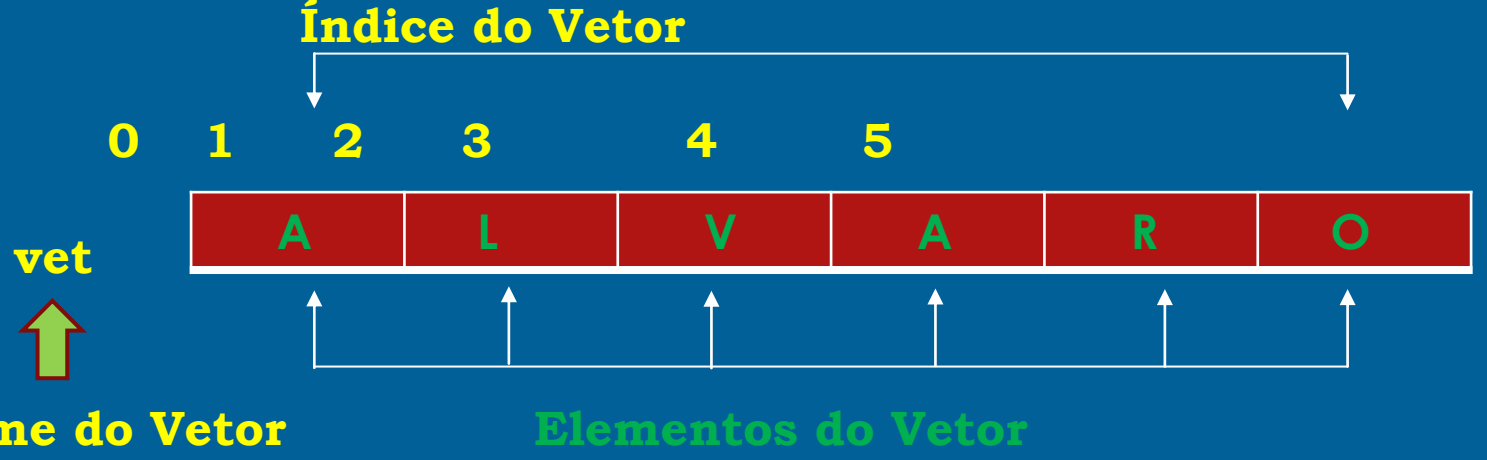
VETOR (ARRAYS)

Exemplo:

int vet[6]; // vetor com 6 elementos do tipo int. Os índices vão de 0 a 5.



char vet[6]; // vetor com 6 elementos do tipo int. Os índices vão de 0 a 5.



VETOR (ARRAYS)

Em diversas situações os tipos básicos de dados (int, float, char, ...) não são suficientes para representar a informação que se deseja armazenar. Exemplo, uma palavra: “AULA”.

Existe a possibilidade de construção de novos tipos de dados a partir da composição (ou abstração) de tipos de dados primitivos. Esses novos tipos têm um formato denominado ESTRUTURA DE DADOS, que define como os tipos primitivos estão organizados.

O vetor é uma estrutura de dados indexada, que pode armazenar uma determinada quantidade de valores do mesmo tipo.

Os dados armazenados em um vetor são chamados de itens do vetor.
Para localizar a posição de um item em um vetor usamos um número inteiro denominado índice do vetor.

Sintaxe:

```
tipo_primitivo identificador[qtde_elementos];
```

Exemplo:

```
int dados[5]; // vetor com 5 (0 a 4) elementos do tipo int
```

Declarando e inicializando de vetores

Exemplo 1:

Podemos declarar e inicializar um vetor com um tamanho constante, como abaixo:

```
int numeros[5] = {10, 20, 30, 40, 50};
char palavra[9] = {‘c’, ‘a’, ‘d’, ‘e’, ‘i’, ‘r’, ‘a’};
```

Exemplo 2:

Iniciando apenas alguns elementos do vetor:

```
int valores[5] = {2,4,6};
```

será equivalente a

```
int valores[5] = {2,4,6,0,0};
```

Isto ocorre porque apenas alguns itens do vetor foram inicializados. Neste caso, quando o número de itens inicializados é menor que o número total de itens do vetor, os itens não inicializados são automaticamente zerados.

Exemplo 3:

Inicializando um vetor sem especificar a quantidade de elementos

```
int valores[] = {3,5,7};
```

EXEMPLO:

```
#include<stdio.h>
main()
{
  float notas[5] = {7, 8, 9.5, 9.9, 5.2}; // declarando e inicializando o vetor notas

  printf ("Exibindo os Valores do Vetor \n\n");
  printf ("notas[0] = %.1f\n", notas[0]);
  printf ("notas[1] = %.1f\n", notas[1]);
  printf ("notas[2] = %.1f\n", notas[2]);
  printf ("notas[3] = %.1f\n", notas[3]);
  printf ("notas[4] = %.1f\n", notas[4]);
 }
```

EXPLICAÇÃO DO CÓDIGO

Para fazer referência a uma determinada posição do vetor, devemos utilizar o nome do array e seu respectivo índice.

Por exemplo:

notas[0], faz referência ao elemento armazenado no vetor notas posição (índice) zero.
Para exibir esse elemento na tela usamos:
printf(“notas[0] = %.1f\n”, notas[0]);

Inserindo Dados no vetor com o Comando While

Exemplo:

```
#include <stdio.h>
main()
{
int vet[4];
int i;

i = 0;
while (i<=3) // rotina para preencher o vetor
      {
         printf("Digite o numero: \n");
         scanf("%d", &vet[i]);
         i++;
      }

i = 0;
while (i<=3) // rotina para exibir o conteúdo do vetor
      {
         printf("Numero %d\n", vet[i]);
         i++;
      }
}
```

Inicializando Vetor:

Com lixo

```
#include <stdio.h>
int main()
{
    int lixo[10];
    int indice = 0;
    while(indice < 10)
        {
            printf("Lixo na posicao %d: %d\n", indice, lixo[indice]);
            indice++; // contador. A cada passagem o indice é incrementado com mais 1
        }
}
```

Sem lixo

```
#include <stdio.h>
int main()
{
    int lixo[10] = {}; // abriu e fechou chave. Inicializou o vetor com zero
    int indice = 0;
    while (indice < 10)
        {
            printf("Lixo na posicao %d: %d\n", indice, lixo[indice]);
            indice++; // contador. A cada passagem o indice é incrementado com mais 1
        }
}
```

Inicializando Vetor e Identificando o fim da Informação no Vetor:

```
#include <stdio.h>
int main()
{
    char palavra[10] = {}; // Inicializou o vetor com zero
    int i = 0;

    printf ("Digite uma Palavra: ");
    scanf ("%s", &palavra);

    while (i < 10)
            {
                if (palavra[i] != '\0')
            printf("Informacao:  %c e indice: %d\n", palavra[i], i);
                else
            printf("Informacao truncada:  %c e indice: %d\n", palavra[i], i);
          i++;
      }
}
```

EXEMPLO:

```
main()
{
char pal[10];
int i;
i = 0;
scanf (“%s”, &pal);
while (i<=10) // (i <=10) é a condição. Quantas vezes a
        estrutura irá repetir
     {
        printf(“O caractere e:  %c", pal[i]); Imprime o
    caractere referente ao índice de i
        printf("Número %d\n", i); Imprime o índice de i
        i++; // contador. A cada passagem o i é
    incrementado com mais 1
      }
}
```

SOMA E CONTAGEM:

```
#include<stdio.h>
main()
{
char pal[10] = {}; // vetor com 10 posições
int i = 0, soma = 0, cont = 0, tam = 0;
float media = 0;

printf("Digite uma Palavra: ");
scanf ("%s", &pal);
tam = strlen(pal);
while (i <= tam-1) // diminui um do valor total da variável tam
      {
         printf("Numero do Indice: %d\n", i);
         soma = soma + i; // faz a soma do indice
         cont++; // Contador. Incrementa de um em um
         i++;
      }
media = float(soma) / cont;
printf("A media eh: %f", media);
}
```

Comando For

O comando for é de alguma maneira encontrado em todas linguagens procedurais de programação.
A condição é uma expressão de relação que testa a variável de controle do loop contra algum valor para determinar quando o loop terminará. O incremento define a maneira como a variável de controle do loop será alterada cada vez que o computador repetir o loop.

Sintaxe:

```
for (inicialização; condição; incremento/decremento)
{
        <comando>;
        <comando>;
        <comando>;
}
```

Exemplo:

```
#include<stdio.h>
main ()
{
int i, idade;
for (i=0; i<=5; i++) // controla a inicialização, a condição e o incremento
    {
        printf (“ Digite uma idade”);
        scanf (“%d”, &idade);
        printf (“A idade digitada foi %d \n”);
    }
}
```

Exemplo:

```
#include<stdio.h>
main ()
{
int i, idade;
for (i = 8; i >= 4; i--) // controla a inicialização, a condição e o decremento
    {
        printf (“ Digite uma idade”);
        scanf (“%d”, &idade);
        printf (“A idade digitada foi %d \n”);
    }
}
```

Quando for trabalhar com o decremento a inicialização e a condição precisam ser revistas para atender a demanda.

Diferença de Código entre While e For


```c
#include <stdio.h>
main ()
{
int i, idade;
i = 0;

while (i<=3)
    {
        printf ("Digite uma idade");
        scanf ("%d", &idade);
        i++;
    }
}
```

```c
#include <stdio.h>
main ()
{
int i, idade;

for (i=0; i<=3; i++)
    {
        printf ("Digite uma idade");
        scanf ("%d", &idade);
    }
}
```

Matrizes (Vetores Multidimencionais)

Como os vetores, as matrizes são estruturas de dados homogêneas. Podem ser construídas dos diversos tipos básicos primitivos (float, int, char).

A Principal diferença em relação aos vetores (unidimensionais): possui uma ou mais dimensões adicionais, mas na maioria dos casos: utiliza-se matrizes bidimensionais.

São utilizadas quando os dados homogêneos necessitam de uma estruturação com mais de uma dimensão.

Exemplos:

Programar um jogo de xadrez (o tabuleiro é naturalmente bidimensional), estrutura para guardar caracteres de um livro (três dimensões: 2 para representar os caracteres de uma página e uma terceira para indicar as páginas),problemas matemáticos matriciais.

A sintaxe para declaração de uma variável deste tipo é semelhante à declaração dos vetores. Considera-se, porém a quantidade de elementos da outra dimensão:

tipo_primitivo identificador[qde_linha] [qde_col];

Exemplo:

int mat[2][3]; uma matriz de números inteiros com 2 linhas e 3 colunas, essa matriz possui 6 elementos.

Neste caso os índices variam de 0 a 2 para linhas e de 0 a 1 para colunas.

		Índices das Colunas		
		0	1	2
Índices das Linhas	0	1.7	4.9	2.8
	1	5.3	9.1	7.4

Exemplo:

float mat[4][2] = {{3.6, 2.7}, {5.0, 4.1}, {7.9, 9.8}, {5.3, 2.4}};

Ou seja, fornecem-se os valores linha a linha e coluna por coluna.

Observações:
As matrizes não são inicializadas automaticamente.

int num[5][3] = {{32, 64, 27}}; // só preenche a primeira linha
Se houver menos valores do que o número de elementos da matriz, os elementos restantes são inicializados automaticamente com o valor zero.

int mat[4][4] = {{0}}; //todos elementos são nulos

A seguinte declaração causa um erro de sintaxe:
int mat[2][5] = {{32, 64, 27, 18, 95, 14}, {12, 15, 43, 17, 67, 31}};

Uma vez que há mais elementos do que espaços de memória alocados. No exemplo acima existem 6 elementos da coluna e a definição da matriz é de 5 coluna.

Os elementos das matrizes são referenciados individualmente por meio de índices (iniciando de zero) entre colchetes.

Criar um algoritmo que leia os elementos de uma matriz inteira de 4 x 4 e imprimir os elementos.

```
#include <stdio.h>
main()
{
int lin,col;
int mat[4][4];

for (lin=0; lin<=3; lin++) // controle da linha
{
	for (col=0; col<=3; col++) // controle da coluna
	{
		printf("Digite o elemento da linha %d, coluna %d da matriz: ", lin+1, col+1);
		scanf("%d", &mat[lin][col]); // armazena a informação na matriz conforme posição de linha e coluna
	}
}
printf("Matriz\n");

for (lin=0; lin<=3; lin++) // controle da linha
{
	for (col=0; col<=3; col++) // controle da coluna
	{
		printf("%d\t", mat[lin][col]); // exibe a informação da matriz conforme posição de linha e coluna
	}
	printf("\n");

}
}
```