Universidade Federal de Juiz de Fora - UFJF
LabMaker - Impressão 3D nas Escolas

Curso de Robótica: Introdução a Linguagem C

Alexandre Vitor Silva Braga
Thaís de Jesus Soares
29 de março de 2023

1. INTRODUÇÃO

A estrutura básica de um programa em C pode ser vista baixo.

```c
#include <stdio.h>
int main (){
    // Bloco de Comandos
    return 0;
}
```

Na qual,

- **stdio.h:** Biblioteca padrão da linguagem C, responsável pela entrada e saída de dados.
- **int:** Tipo de retorno da função.
- **main:** Função principal do programa, a partir da qual o código começa a ser executado.
- **Duas Barras Invertidas (//):** Utilizada para inserir comentários no código.
- **Ponto e Vírgula (;):** Empregada para finalizar uma linha de comando.
- **return:** Comando após o qual estão dispostos os parâmetros de retorno da função.

2. VARIÁVEIS E CONSTANTES

Em um programa, um valor pode ser classificado como:

- **Variável:** Dado cujo valor pode variar ao longo da execução do programa.
- **Constante:** Dado cujo valor se mantém fixo ao longo da execução do programa.

Ainda, torna-se relevante salientar que, para ambos os casos, é necessário seguir algumas regras ao nomear esses parâmetros. A saber, o nome:

- Deve ser único, ou seja, não é possível declarar dois ou mais parâmetros com mesma nomenclatura;
- Não pode ser uma palavra já reservada da linguagem C. Ex: **int**, **main** e **return**.
- Pode conter apenas letras, dígitos e sublinha. Ex: *variavel_1*.
- Deve começar com uma letra. Para variáveis costuma-se usar minusculas. Ex: *largura*. Por outro lado, para constantes, costuma-se escrever todas as letras em maiúsculo. Ex: *PI*.
- Deve permitir a identificação do valor que representa. Ex: *raio* e *altura* para armazenar as dimensões de um cilindro.

Ademais, ressalta-se que, toda variável e toda constante, possuem um tipo de dados associado. Em C, os tipos principais podem ser vistos abaixo.

- **int:** é capaz de representar números inteiros. Ex: 1, -2 e 30.
- **float:** é capaz de representar números reais (também conhecidos como pontos flutuantes). Ex: -1.0, 2.43 e 30.789.
- **char:** é capaz de representar um único caractere (letra, simbolo, digito e etc). Ex: 'a', '10' e '!'.

2.1 VARIÁVEIS

- **Declaração de Variáveis:** Momento em que a variável é criada no programa, sendo preciso indicar o tipo e o identificador da mesma.

```c
# include <stdio.h>
int main()
{
  // Declaracao da Variavel:
  float raio;
  return 0;
}
```

No exemplo acima, a variável criada tem tipo **float** e identificador *raio*.

- **Inicialização de Variáveis:** Momento em que uma variável tem seu valor inicializado. Antes de tal procedimento o valor contido pela variável é lixo de memória e, portanto, não deve ser usado.

```c
# include <stdio.h>
int main()
{
  // Declaracao da Variavel:
  float raio;
  // Inicializa  o da Variavel:
  raio = 5.0;
  // Alternativa: Declaracao e Inicializacao realizadas de forma conjunta.
  // float raio = 5;
  return 0;
}
```

- **Operador de Atribuição:** Utilizado para modificar o valor de uma variável.

```c
# include <stdio.h>
int main()
{
  // Declaracao da Variavel:
  float raio;
  // Inicializa  o da Variavel:
  raio = 5.0;
  // Alternativa: Declaracao e Inicializacao realizadas de forma conjunta.
  // float raio = 5.0;
  // Comando de Atribuicao:
  raio = raio + 1.0;
  return 0;
}
```

2.2 CONSTANTE

- **Definição de uma Constante:**

```c
# include <stdio.h>
// Definicao da constante:
# define PI 3.14159
int main()
{
  // Declaracao da Variavel:
  float raio;
  // Inicializa  o da Variavel:
  raio = 5.0;
  // Alternativa: Declaracao e Inicializacao realizadas de forma conjunta.
  // float raio = 5.0;
  // Comando de Atribuicao:
  raio = raio + 1.0;
  return 0;
}
```

2.3 EXPRESSÕES

Expressões são combinações de variáveis, constantes e operadores que, quando avaliadas, resultam em um:

- Um número, o qual pode ser inteiro ou real.

```c
# include <stdio.h>
// Definicao da constante:
# define PI 3.14159
int main()
{
  // Declaracao da Variavel:
  float raio;
  // Inicializa  o da Variavel:
  raio = 5.0;
  // Alternativa: Declaracao e Inicializacao realizadas de forma conjunta.
  // float raio = 5.0;
  // Comando de Atribuicao:
  raio = raio + 1.0;
  // Expressao Aritmetica:
  float comprimento = 2 * PI * raio;
  return 0;
}
```

- Verdadeiro ou falso.

A primeira é denominada expressão aritmética, enquanto a segunda é conhecida como expressão lógica. Os operadores de ambas podem ser visualizados nas Tabelas 1, 2 e 3.

Operadores	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Resto

Tabela 1. Operadores usados em expressões aritméticas

Operadores	Operação
==	Igualdade
!=	Diferença
<	Menor
<=	Menor ou igual
>	Maior
>=	Maior ou igual

Tabela 2. Operadores Relacionais usados em expressões lógicas.

!	Negação (NOT)
&&	Conjunção (AND)
||	Disjunção (OR)

Tabela 3. Operadores Lógicos usados em expressões lógicas.

Por fim, é relevante destacar a ordem de prioridade para execução dos operadores em uma expressão:

1. Parênteses (dos mais internos para os mais externos).
2. Expressões aritméticas, seguindo a ordem: funções, * e /, + e -.
3. Expressões lógicas relacionais: <, <=, ==, >, >= e !=.
4. Expressões lógicas, seguindo a ordem: !, &&, ||.
5. Da esquerda para a direita, quando houver indeterminações.

Exercícios:

1. Construa uma sequência de instruções para calcular o volume de um copo com 12 cm de altura e 6 cm de diâmetro, da seguinte forma:
   - Declare as variáveis para raio, altura e volume.
   - Inicialize as variáveis cujo valor é conhecido.
   - Calcule o valor do volume e armazene-o na variável.
2. Construa uma sequência de instruções para indicar quantos dias, horas, minutos e segundos equivalem a 200.000 segundos. Assim como no exercício anterior, declare as variáveis, inicialize-as e, por fim, realize o cálculo armazenando o resultado.

3. ENTRADA E SAÍDA DE DADOS

Ao programar é possível, através do dispositivo de saída padrão (tela) e do dispositivo de entrada padrão (teclado) do computador, realizar operações de entrada e saída dos dados.

Nesse contexto, destacam-se duas funções úteis:

- **scanf:** é responsável pela entrada de dados. Sua sintaxe é da forma:

```
scanf ("Codigos de Formatacao", Lista de Enderecos de Variaveis);
```

O primeiro parâmetro indica o tipo e a ordem esperada dos valores que serão lidos.

Já os posteriores contém uma sequência de variáveis dos tipos indicados nos códigos de formatação, todas precedidas por &, sendo que cada expressão &NomeDaVariavel retorna o endereço da respectiva variável na memória. Isto permite que o programa saiba onde deve ser armazenado o valor lido do teclado.

```c
# include <stdio.h>
// Definicao da constante:
# define PI 3.14159
int main()
{
  // Declaracao da Variavel:
  float raio;
  // Leitura de Dados:
  scanf("\%f", &raio);
  // Expressao Aritmetica:
  float comprimento = 2 * PI * raio;
  return 0;
}
```

- **printf:** é responsável pela saída de dados. Sua sintaxe é da forma:

```c
printf ("Texto + Codigos de Formatacao", Lista de Argumentos);
```

O primeiro parâmetro pode conter mensagens a serem exibidas e códigos de formatação que indicam como o conteúdo de uma variável deve ser exibido.

Já os posteriores, de uso opcional, indica quais valores (identificadores de variáveis, expressões aritméticas e constantes) devem ser impressos. Esses valores devem estar dispostos em uma lista separados por vírgulas.

```c
# include <stdio.h>
// Definicao da constante:
# define PI 3.14159
int main()
{
  // Declaracao da Variavel:
  float raio;
  //Impressao na Tela
  printf("Digite o raio: ");
  // Leitura de Dados:
  scanf("\%f", &raio);
  // Expressao Aritmetica:
  float comprimento = 2 * PI * raio;
  // Impressao na Tela
  printf("Comprimento = \%f", comprimento);
  return 0;
}
```

Na Tabela 4, estão expostos os códigos de formatação para os diferentes tipos de dados.

Código	Tipo	Elemento Armazenado
% c	char	Um caracter
% d	int	Um inteiro
% f	float	Um real
% s	tipo composto	Uma cadeia de caracteres

Tabela 4. Impressão de tipos de Dados.

Exercícios

1. Elabore um programa completo que imprima o dobro, o triplo e o quadrado do valor x. O valor de x deve ser informado pelo usuário através do teclado. Ainda, supondo que o valor lido é 3, a saída de seu programa deve ser:

   - Valor de x: 3

   - Dobro de x: 6

- Triplo de x: 9

- Quadrado de x: 9

Use variáveis para armazenar os valores numéricos que deverão ser impressos.

2. Reescreva o programa ao lado usando apenas duas variáveis.

```c
# include <stdio.h>
int main() {
    float lado1,lado2,lado3;
    float perimetro;
    printf("TRIANGULO\n");
    printf("Digite os lados:");
    scanf("%f%f%f", &lado1, &lado2, &lado3);

    perimetro = lado1+lado2+lado3;
    printf("Perimetro: %f",perimetro);
    return 0;
}
```

3. **Desafio:** Uma empresa contratou um médico para avaliar todos os seus funcionários na própria sede da empresa. Para que cada funcionário saiba o horário agendado para sua consulta médica, você deverá fazer um programa que lê a matrícula do funcionário e informa o dia e horário da consulta. Observe que:

   - As matrículas dos funcionários são números consecutivos entre 1 e 30 (inclusive). Os funcionários serão atendidos em ordem crescente de matrícula.
   - As consultas duram uma hora e serão realizadas em uma única semana, de 2a a 6a. O médico estará disponível das 8 às 14h.

   Para a matrícula 24, por exemplo, o programa deverá imprimir a saída: 5a-feira as 13:00 horas.

4. FUNÇÕES

Uma função é um trecho de código computacional que realiza uma tarefa bem definida. Nesse contexto, pode receber dados e/ou retornar um resultado ao ser executada.

- **Declaração da Função:** especifica a sintaxe dessa. É válido salientar que além do nome, a sintaxe é capaz de especificar os parâmetros (dados que a função precisa receber para executar) e o retorno (o tipo de resultado produzido pela função).

```c
tipo_de_retorno nome_da_funcao (parametros);
```

- **Definição da Função:** é a implementação da mesma. Além de conter o cabeçalho que especifica sua sintaxe (como na declaração), contém a sequência de instruções necessárias para realizar a tarefa.

```c
tipo_de_retorno nome_da_funcao (parametros){
    // Bloco de Comandos
}
```

- **Chamada da Função:** é a utilização da mesma dentro de alguma outra função ou programa. Nesse caso, é necessário indicar o valor de cada informação necessária para executar a função.

```c
nome_da_funcao (parametros);
```

- **Escopo da Variável:** são as linhas do código onde a variável pode ser acessada, lida e/ou modificada.

  Nesse contexto, vale destacar a existência de variáveis locais, declaradas e/ou recebidas como parâmetros de uma função, cujo escopo corresponde apenas ao bloco de comandos dessa. Para exemplificar, tem-se no exemplo abaixo,

as variáveis *raio* e *comprimento*, que só podem ser acessadas de dentro da função **main** e as variáveis *r* e *c*, que só podem ser acessadas de dentro da função *retorna_comprimento*.

Ademais, na chamada da função *retorna_comprimento*, a variável *raio*, passada como parâmetros da função, não possui relação com *r*, uma vez que as duas apontam para endereços de memórias distintos. Ou seja, caso o valor de *r* seja alterado, o valor de *raio* continuará o mesmo. Tal fenômeno é conhecido como **Passagem de Parâmetro por Valor**.

Essas ideias, podem ser melhor vistas na Tabela abaixo, na qual o valor assumido por uma variável durante a execução de uma dada linha do código pode ser visualizado. É o chamado **Teste de Mesa**.

```c
# include <stdio.h>
// Definicao da constante:
# define PI 3.14159
// Declaracao e Definicao da Funcao
float retorna_comprimento (float r){
    float c = 2 * PI * r;
    return c;
}
int main()
{
  // Declaracao da Variavel:
  float raio;
  //Impressao na Tela
  printf("Digite o raio: ");
  // Leitura de Dados:
  scanf("\%f", &raio);
  // Chamada da Funcao
  float comprimento = retorna_comprimento(raio);
  // Impressao na Tela
  printf("Comprimento = \%f", comprimento);
  return 0;
}
```

Linha	raio	comprimento	r	c
12	?	?	?	?
16	6	?	?	?
18	6	?	?	?
6	6	?	6	37.7
18	6	37.7	?	?

No que tange as funções, ressalta-se ainda que essas estão sujeitas as mesmas regras aplicadas ao nomear variáveis. E, caso não produzam um valor como resultado devem ter tipo de retorno informado como **void**.

Exercícios:

1. a) Escreva uma função que recebe dois números inteiros e imprime a soma, o produto, a diferença, o quociente e o resto entre esses dois números.

   b) Faça um programa em C (função principal) que leia dois inteiros do teclado e chame a função da letra a).

   c) Teste seu programa com os valores 11 e 3.

2. Realize o teste de mesa para o código abaixo.

```c
#include <stdio.h>
void imprimeFormatoHora (int qtddMinutos)
{
    int hora, min;
    hora = qtddMinutos / 60;
    min = qtddMinutos % 60;
    printf("%02d:%02d", hora, min);
    return;
}
int main()
{
    int minutos;
    printf("Entre com os minutos: ");
    scanf("%d",&minutos);
    printf("%d minutos equivale a ",minutos);
    imprimeFormatoHora(minutos);
    return 0;
}
```

3. Considerando a fórmula para o cálculo da distância entre dois pontos (x1, y1) e (x2, y2):

   a) Escreva uma função que receba como parâmetros as coordenadas de dois pontos e retorne a distância entre eles.

   b) Escreva um programa em C (função principal) que capture do teclado as coordenadas dos 3 vértices de um triângulo, calcule e imprima o perímetro deste triângulo, chamando a função anterior.

   c) Teste seu programa, simulando sua execução com as seguintes coordenadas: (4,1), (1,1), (4,5).

4. **Desafio:** Faça um programa para calcular quantas latas de verniz serão necessárias para cobrir um deque de madeira.

   O usuário do programa informará a largura e o comprimento da superfície a ser coberta e o programa deverá imprimir o número de latas necessárias (valor inteiro), dado que cada lata de verniz cobre até 3 m^2 de superfície.

   O programa deverá ter no mínimo 3 funções. Teste o programa calculando o necessário para cobrir uma superfície de 4.5 x 5m.

   Tente identificar as tarefas que poderão constituir diferentes funções e, para cada tarefa, especifique os dados de entrada (parâmetros) necessários para sua execução e defina se esta tarefa produzirá ou não um resultado (retorno).

5. ESTRUTURAS DE CONTROLE CONDICIONAL

As estruturas de controle condicional permitem que um mesmo programa se comporte de forma distinta ante situações distintas. Já as expressões lógicas, apresentadas anteriormente, são as responsáveis por identificar tais situações. Isso é feito, ao verificar se o valor de uma ou mais variáveis satisfaz uma determinada condição durante a execução do programa e retornar, como resultado, verdadeiro ou falso.

Nesse sentido, torna-se válido ressaltar que, em C, o valor zero é interpretado como falso e, apesar de qualquer valor diferente de zero ser considerado verdadeiro, o mais comum é o uso do valor 1.

- **Operadores Relacionais (Tabela 2):** Comparam dois valores, retornando 1 ou 0 como resultado. Para exemplificar, considere o código abaixo e seu respectivo teste de mesa.

```c
#include <stdio.h>
int main (){
    // Declaracao e Inicializacao
    // das variaveis
    int idade = 18;
    // resultado = 1, se idade for
    // menor que 18
    int resultado = idade < 18;
    // resultado = 1, se idade for
    // maior ou igual a 60
    resultado = idade >= 60;
    // resultado = 1, se idade
    // for igual a 18
    resultado = idade == 18;
    return 0;
}
```

Linha	idade	resultado
5	18	?
8	18	0
11	18	0
14	18	1

- **Operadores Lógicos (Tabela 3):** Combinam expressões lógicas. Os resultados para tais combinações, dado duas condições *a* e *b* podem ser vistos na Tabela 5), também conhecida como Tabela Verdade.

a	b	!a	!b	a && b	a || b
V	V	F	F	V	V
V	F	F	V	F	V
F	V	V	F	F	V
F	F	V	V	F	F

Tabela 5. Tabela Verdade.

Observações:

1. Na Tabela 5, nota-se que um expressão com operador **E** é falsa, se ao menos um operando é falso. Já uma operação com **OU** é verdadeira, se ao menos um operando é verdadeiro.

2. Operadores relacionais precisam de operadores lógicos para serem combinados.

- **Estruturas condicionais:** Possuem sintaxe conforme descrito abaixo, sendo que as instruções contidas no "Bloco de Comandos"são executados somente quando *condicao* é verdadeira. Caso contrário executa-se as contidas em "Bloco de Comandos Alternativo".

```c
if (condicao){
    // Bloco de Comandos
}
else {
   // Bloco de Comandos Alternativo
}
```

```c
#include <stdio.h>
int main (){
    // Declaracao das Variaveis
    int quantidade;
    float precoUnitario, preco;
    // Leituras dos Dados
    scanf ("%d%f", &quantidade, &precoUnitario);
    // Inicializacao da Variavel preco
    preco = quantidade * precoUnitario;
    // Verificacao: Se quantidade > 10 E precoUnitario > 50
    // entra no bloco de comandos dentro do if
    if (quantidade > 10 && precoUnitario > 50.0 )
    {
        preco = preco * 0.85;
        printf ("Ganhou um desconto de 15%%!\n");
    }
    //Sen o entra dentro do bloco de comandos do else
    else
    {
        preco = preco * 0.95;
        printf ("Toda a loja com 5%% de desconto!\n");
    }
    // Impress o do Dado
    printf ("\nPreco final: %f", preco);
}
```

Exercícios:

1. Construa a função calculaPesoIdeal que recebe o sexo e a altura de uma pessoa como parâmetros. A função deve calcular e retornar o peso ideal da pessoa, utilizando uma das seguintes fórmulas:

   - masculino: (72.7 * alt) - 58;
   - feminino: (62.1 * alt) - 44.7.

   Faça um programa que leia o sexo, a altura e o peso de uma pessoa e imprima se esta pessoa está acima, abaixo ou com o peso ideal. Teste seu programa com os valores F, 1.71 e 59.5.

2. Elabore um programa que leia 3 valores reais *(x, y e z)* de comprimento e imprima na tela se tais valores formam os lados de um triângulo ou não. Para formar um triângulo, os valores devem atender às seguintes condições:

   $x < y + z$ e $y < x + z$ e $z < x + y$

   Teste seu programa com os valores 4, 2.2 e 1.4.

3. Para auxiliar os vendedores de uma loja na orientação aos clientes sobre as diversas formas de pagamento, desenvolver um algoritmo para:

   (a) Imprimir o menu abaixo.
   Forma de pagamento:
   1. À vista.
   2. Em duas vezes.
   3. Em três vezes.
   4. A partir de quatro vezes.

   Entre com sua opção:

   (b) Ler o código da opção de pagamento.

   (c) Imprimir uma das mensagens de acordo com a opção lida:
   Opção = 1: Desconto de 20%
   Opção = 2 ou 3: Mesmo preço a vista
   Opção = 4: Juros de 3% ao mês
   Opção < 1 ou opção > 4: Opção inválida

6. ESTRUTURAS DE CONTROLE DE REPETIÇÃO

Observe o código abaixo com as estruturas condicionais que acabamos de aprender. Ele cumpre sua função de imprimir certo número de vezes uma mensagem de acordo com o número de vezes desejado.

```c
#include <stdio.h>
int main()
{
    int numVezes;
    printf("Quantas vezes a impressao deve ser feita?");
    scanf("%d", &numVezes);
    if( numVezes == 1 )
        printf("imprima essa mensagem\n");
    else
        if( numVezes == 2 ){
            printf("imprima essa mensagem\n");
            printf("imprima essa mensagem\n");
        }
        else
            if( numVezes == 3 ){
                printf("imprima essa mensagem\n");
                printf("imprima essa mensagem\n");
                printf("imprima essa mensagem\n");
            }
            else
                if( numVezes == 5 ){
                    printf("imprima essa mensagem\n");
                    printf("imprima essa mensagem\n");
                    printf("imprima essa mensagem\n");
                    printf("imprima essa mensagem\n");
                    printf("imprima essa mensagem\n");
                }
    return 0;
}
```

Mas existe um porém. E se fosse necessário imprimir muitas vezes ? 20 ? 50 ? 100 ? Muito código seria repetido desnecessariamente.

É importante se atentar a 2 perguntas principais:

- **Qual** comando precisa ser repetido ?
- **Quantas** vezes ele precisa ser repetido ?

6.1 ESTRUTURAS DE REPETIÇÃO

- **Teste no Início** (While): Se a condição for verdadeira, o bloco é executado e a condição é verificada novamente. Se a condição for falsa, o bloco é ignorado e o algoritmo segue adiante

```
inicializacao
while(condicao){
    comandos1
    atualizacao
}
comandos2
```

- **Teste no Final** (Do While): O bloco é executado pelo menos uma vez e só depois a condição é verificada. Se a condição for verdadeira, o bloco é executado novamente e assim por diante.

```
inicializacao
do {
    comandos1
    atualizacao
} while (condicao);
comandos2
```

- **Variável de Controle** (For): Utilizam uma variável que controla quantas vezes o bloco será executado. A variável tem um valor inicial, um valor final e é modificada dentro do laço de repetição. O bloco é executado enquanto a variável estiver dentro do intervalo definido.

```
for (inicializacao; condicao; atualizacao)
{
    blocoDeComandos1;
}
blocoDeComandos2;
```

Estruturas de repetição sempre seguem um padrão, ao qual deve-se atentar:

```
void imprimePontosNoCampeonato(int numTimes)
{
    int cont, vitorias, empates, derrotas;
    printf("\nPara cada time, digite o numero ");
    printf("de vitorias, empates e derrotas: ");
    cont = 0; //Inicializacao
    while( cont < numTimes ){ //Condicao
        printf("\nTime %d: ", cont + 1);
        scanf("%d%d%d", &vitorias, &empates, &derrotas);
        printf(" Total: %d pontos", (3 * vitorias) + (1 * empates)
            );
        cont++; //Atualizacao
    }
}
```

- Condição: Teste envolve normalmente ao menos uma variável.
- Inicialização: Toda variável precisa ser inicializada antes do laço através de atribuição ou leitura.
- Atualização: Ao menos uma variável da condição precisa ser atualizada no interior do laço.

6.2 ACUMULADORES

Em muitos casos como por exemplo a soma de uma sequência, desejamos ser capazes de armazenar um valor acumulado, ou realizar diferentes ações repetidas vezes.
Para isso pode-se utilizar um **Acumulador**. Uma variável a qual recebe um acréscimo a cada iteração, ou seja, a cada atualização de valor, um valor variável é somado ao valor anterior. soma = soma + novo_valor; O que significa, que o valor que soma receberá atualmente, é o valor que possuía, adicionada de um novo valor.

Acumuladores podem ser utilizados como contadores, que aumentam ou diminuem constantemente para contagem de ocorrências. Já variáveis de controle também podem ser contadores ou acumuladores. A seguir, um exemplo de acumulador de soma de números inteiros:

```c
#include <stdio.h>
using namespace std;
int main()
{
    int num, soma;
    soma = 0; // inicializa acumulador
    printf("Digite um numero inteiro: ");
    scanf("%d", &num);
    while( num != 0 )
    {
        soma = soma + num; //atualiza acumulador
        printf("Soma parcial: %d",soma);
        printf("\nDigite um numero inteiro: ");
        scanf("%d", &num);
    }
    printf("Soma total: %d",soma);
    return 0;
}
```

6.3 TESTE DE MESA

```c
#include <stdio.h>
int main()
{
    int num, soma;
    soma = 0; // inicializa acumulador
    printf("Digite um numero inteiro: ");
    scanf("%d", &num);
    while( num != 0 )
    {
        soma = soma + num; //atualiza acumulador
        printf("Soma parcial: %d",soma);
        printf("\nDigite um numero inteiro: ");
        scanf("%d", &num);
    }
    printf("Soma total: %d",soma);
    return 0;
}
```

linha	num	soma	teste
4	?	?	
5	?	0	
7	9	0	
8	9	0	V
10	9	9	
13	-2	9	
8	-2	9	V
10	-2	7	
13	0	7	
8	0	7	F
15	0	7	

6.4 INTERRUPÇÕES (BREAK)

Antes de partir para o fim da seção e seus exercícios, uma última coisa precisa ser vista, o comando **break**. Ele faz com que o fluxo de execução saia do laço de repetição e a execução continue logo após o fim do laço. Ele é extremamente útil para encontrar exceções no código, ou evitar verificações desnecessárias caso o resultado já tenha sido encontrado.

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

int main() {
    int n; // numero a ser verificado
    int i; // variavel auxiliar para o la o
    int raiz; // raiz quadrada de n
    int primo = 1; // vari vel que indica se n    primo ou n o

    printf("Digite um n mero inteiro positivo: ");
    scanf("%d", &n); // le o numero do usuario

    if (n <= 1) // se o n mero for menor ou igual a 1
    {
        printf("O n mero %d nao e primo.\n", n); // ele nao e primo
        return 0; // encerra o programa
    }

    raiz = (int) sqrt(n); // calcula a raiz quadrada de n

    // percorre os possiveis divisores de n entre 2 e raiz
    for (i = 2; i <= raiz; i++) {
        if (n % i == 0) // se n for divisivel por i
        {
            primo = 0; // n nao e primo
            break; // sai do laco
        }
    }

    if (primo == 1) { // se n for primo
        printf("O n mero %d e primo.\n", n);
    } else { // se n n o for primo
        printf("O n mero %d nao e primo.\n", n);
    }

    return 0;
}
```

Exercícios:

1. Como poderia ser reescrito o código do início da seção utilizando uma estrutura de repetição ?

2. Faça um programa que imprime todos os números pares de 1 a 100.

3. Tente fazer o código para imprimir números primos do zero.

4. **Desafio**: Faça um programa que consiga "adivinhar"um número de 1 a 100 que o usuário pensar com até 7 perguntas. As perguntas devem ser do tipo:

    - "O número é maior que 10 e menos ou igual a 20 ?"
    - "O número é 83 ?".
    - O usuário só poderá responder "S"(sim) ou "N"(não).

7. VETORES

Imagine que você precisa criar um programa que lê as notas de 4 alunos e calcula a sua média. Isso pode ser feito facilmente com os conhecimentos adquiridos nas últimas seções, utilizando entradas de dados, estruturas de repetição.

```c
int main()
{
    int i;
    float nota, media, soma = 0;
    for (i = 0; i < 4; i++)
    {
        printf("Digite uma nota:");
        scanf ("%f", &nota);
        soma += nota;
    }
    media = soma / 4;
    printf("Media = %f", media);
    return 0;
}
```

O código é funcional, mas e se fosse necessário imprimir o número de notas acima da média ? Seria preciso realizar modificações. A seguir, a primeira tentativa desse código modificado:

```c
// Opcao 1: le duas vezes cada valor
int main()
{
    int i, cont=0;
    float nota, media, soma = 0;
    for (i = 0; i < 4; i++){
        printf("Digite uma nota:");
        scanf ("%f", &nota);
        soma += nota;
    }
    media = soma / 4;
    printf("Digite tudo de novo!");
    for (i = 0; i < 4; i++){
        printf("Digite uma nota:");
        scanf ("%f", &nota);
        if( nota > media )
            cont++;
    }
    printf("Media = %f", media);
    printf("%d notas acima", cont);
    return 0;
}
```

A primeira vista parece um código funcional, mas e se o número aumentasse para uma turma inteira de 30 alunos ? Ou se ainda cada um tivesse notas de 3 provas, resultando em 90 notas ? Não seria legal ter que redigitar os valores todas as vezes. E se fosse utilizada uma variável por nota ? Não seria necessário redigitar valores.

```c
//Opcao 2: uma variavel por nota
int main()
{
    int i, cont=0;
    float n1, n2, n3, n4, n5,
    n6, n7, n8, n9, n10;
    float media, soma = 0;
    printf("Digite as notas:");
    scanf ("%f %f %f %f %f", &n1, &n2, &n3, &n4, &n5);
    scanf ("%f %f %f %f %f", &n6, &n7, &n8, &n9, &n10);
    media = (n1 + n2 + n3 + n4 + n5 + n6 + n7 + n8 + n9 + n10) / 10;
    printf("Media = %f", media);
    if( n1 > media ) cont++;
    if( n2 > media ) cont++;
    if( n3 > media ) cont++;
    if( n4 > media ) cont++;
    if( n5 > media ) cont++;
    if( n6 > media ) cont++;
    if( n7 > media ) cont++;
    if( n8 > media ) cont++;
    if( n9 > media ) cont++;
    if( n10 > media ) cont++;
    printf("%d notas acima", cont);
    return 0;
}
```

Porém com uma grande quantidade de variáveis, torna-se mais fácil cometer erros ao replicá-las. Imagine ter de replicar 100 ou 1000 variáveis, seria um problema. É feita necessária uma estrutura que:

- Permita armazenar uma sequência de dados de um mesmo tipo;
- Permita que dados sejam armazenados e estruturados de forma simples.
- Permite que cada um dos dados armazenados seja acessado diretamente.

A estrutura que nos permite tudo isso é denominada **Vetor**

Ao criar um vetor deve-se sempre indicar seu tamanho, ou seja, quantos valores quer armazenar nele. Como o computador não tem memória infinita, antes de iniciar a execução, é preciso saber quanto espaço de memória necessita ser separado. E isto deve ser especificado através de um **valor constante**
A sintaxe para criação de um vetor é

```c
tipo nomeDoVetor[tamanho];
```

Para por exemplo um vetor de dados quaisquer:

```c
int dados[5];
```

Já por exemplo a criação de vetores para o programa de calcular notas desejado

```c
int main()
{
    int matricula[50]; //matricula de cada aluno
    float notaP1[50]; //nota de cada aluno na 1a prova
    float notaP2[50]; //nota de cada aluno na 2a prova
    float notaP3[50]; //nota de cada aluno na 3a prova
    float mediaTurma[3];//media das notas em cada prova
    ...
}
```

7.1 ACESSANDO VETORES

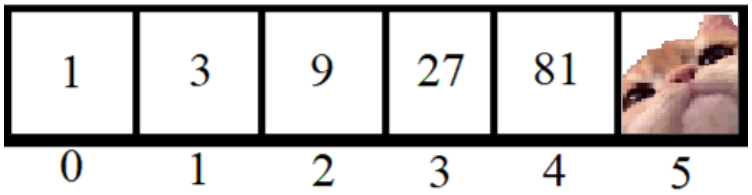
Para permitir esse acesso individual, a sequência de elementos no vetor é sempre numerada, assim quando quiser acessar um elemento específico, basta indicar seu índice. O acesso ao elemento então é feito com:

```
nomeDoVetor [indice]
```

Deve-se ter bastante atenção à numeração de um Vetor. Pois, não é possível fazer uma operação com todo o conjunto dos dados armazenados no vetor de uma só vez!

- O primeiro elemento do vetor sempre recebe o índice 0"(zero).
- Em um vetor com N elementos, os índices portanto são numerados de "0"a "N-1".
- Em algumas linguagens de programação o primeiro índice pode ser "1", por isso é sempre bom estar atento à linguagem que está sendo utilizada. Como estamos utilizando *C*, o índice é "0".

```
int main(){
    int potencias[5]; //vetor com 5 elementos
    potencias[0] = 1;
    scanf("%d", &potencias[1]);
    potencias[2] = pow(potencias[1],2);
    potencias[3] = pow(potencias[1],3);
    potencias[4] = pow(potencias[1],4);
    printf("%d^%d=%d", potencias[1], 0, potencias[0]);
    printf("%d^%d=%d", potencias[1], 1, potencias[1]);
    printf("%d^%d=%d", potencias[1], 2, potencias[2]);
    printf("%d^%d=%d", potencias[1], 3, potencias[3]);
    printf("%d^%d=%d", potencias[1], 4, potencias[4]);
}
```



O que aconteceria se tentassemos acessar o índice potencias[5] por engano ? O que teríamos lá ?

7.2 PASSAGEM POR REFERÊNCIA

Em C, vetores são passados sempre por referência. Ou seja, as modificações feitas no vetor que é um parâmetro no interior de uma função refletem nos dados do vetor passado como parâmetro pela função chamadora.

```c
#define TAMANHO 10
// definicao de outras fun   es
// leVetor, imprimeVetor, maiorElemento
float mediaVetor(int vet[], int tam)
{
    int i, soma = 0;
    for(i = 0; i < tam; i++){
        soma = soma + vet[i];
    }
    return soma / (float)tam;
}
int main()
{
    int v[TAMANHO];
    leVetor(v, TAMANHO);
    imprimeVetor(v, TAMANHO);
    printf("\nMaior = %d.", maiorElemento(v, TAMANHO));
    printf("\nM dia = %.2f.", mediaVetor(v, TAMANHO));
    return 0;
}
```

Na função acima, ao passarmos o vetor "v"para a função *mediaVetor()*, o acesso é feito diretamente onde o vetor "v"está armazenado na memória.

Exercícios:

1. Leia um vetor de 10 posições (inteiros) e imprima-o na ordem invertida (da última para a primeira posição).

2. Faça um programa que preencha um vetor de elementos inteiros com valores lidos do teclado e, ao final, imprima somente valores ímpares armazenados nos índices pares.

3. Faça um programa que preenche um vetor de inteiros com valores lidos do teclado. O programa deve verificar se há elementos repetidos no vetor e imprimir os índices de todos os pares de elementos repetidos.

4. **Desafio** - Faça um programa que apague um elemento no meio de um vetor, de forma que os elementos posteriores à posição deletada sejam deslocados para a esquerda.

8. STRINGS

Sequências de caracteres são essenciais ao fazer programas computacionais. Com eles é possível fazer:

- Mensagens de uma conversa;
- Textos de um programa;
- Nome e endereço em cadastro de clientes de uma loja;
- E até mesmo uma Sequencia genética. Um gene (ou o DNA de algum organismo) é composto de sequencias dos caracteres A, T, G e C (nucleotídeos);

Relembrando, a atribuição e immpressão de caracteres é feita da seguinte forma:

```c
char c = 'a';
printf("int : %d char: %c\n", c, c);
//int : 97 char : a
```

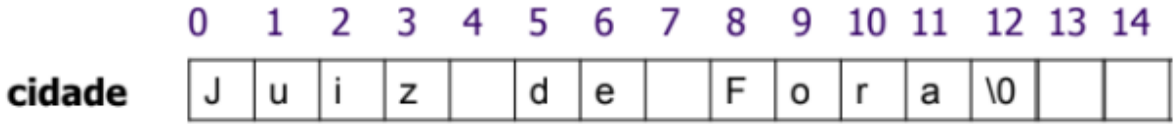
Se com uma variável do tipo char armazena-se somente um caractere, então para armazenar vários caracteres, é necessário utilizar as sequências de caracteres, representadas por vetores do tipo char. Essas sequências são conhecidas como **String**, e obrigatoriamente, sempre terminadas pelo caractere nulo: '\0' (\zero).

String Juiz de Fora

```
1 char cidade[15] = {'J','u','i','z',' ','d','e',' ','F','o','r','a','\0'};
```

ou então de forma mais compacta:

```
1 char cidade[15] = "Juiz de Fora";
```



8.1 SCANF PARA STRINGS

- A leitura será feita até encontrar um caractere branco - espaço (' '), tabulação ('̂') ou nova linha (") – ou chegar ao tamanho máximo indicado no %s.
- Se digitar "Rio de Janeiro"por exemplo, a string "s"conterá apenas "Rio";
- Caso deseje ignorar esses caracteres, deve-se usar:

```
1 scanf(" %c", &ch);
```

```
1 int main()
2 {
3     char s[8];
4     printf("Digite uma string: ");
5     scanf("%7s", s);
6     printf("String digitada: %s", s);
7     return 0;
8 }
```

8.2 BIBLIOTECA DE STRINGS

Existem várias funções em C para manipulação de strings.

- Essas funções estão no arquivo string.h.
- strcpy(char destino[], char origem[]): copia a string origem na string destino
- strlen(char str[]): retorna o tamanho da string "str"
- strcat(char destino[], char origem[]): faz concatenação (junção) da string origem com a string destino. O resultado é armazenado na string destino

Exercícios:

1. Fazer um programa para contar o número de vogais numa cadeia de caractere.
2. Criar uma função para verificar se a string s2 está contida na string s1. A função deverá retornar 1 se encontrar a

string ou 0, caso contrário.

3. Escrever um programa que leia uma string (com mais de uma palavra) e faça com que a primeira letra de cada palavra fique em maiúscula.

4. Fazer um programa para criar e imprimir uma string que será a concatenação de duas outras strings lidas.

5. **Desafio** - Um dos sistemas de encriptação mais antigos é atribuído a Júlio César:

   - Se uma letra a ser criptografada é a letra de número N do alfabeto, substitua-a com a letra (N+K), onde K é um número inteiro constante (César utilizava K = 3).
   - Dessa forma, para K = 1 a mensagem "Adoro programar em C" se torna "Bepsp!qsphsbnbs!fn!D".

   Faça um programa que receba como entrada uma mensagem e um valor de K e altere a mensagem criptografando-a pelo código de César.

9. MATRIZES

Relembrando da seção 7, vetores são estruturas homogêneas, isto é, que permitem a sequência de dados de apenas um tipo. E assim como os vetores, as **Matrizes** também são estruturas de dados homogêneas. Podendo ser construídas dos diversos tipos básicos primitivos (real, inteiro, caractere). Porém possuem uma diferença principal em relação aos seus parentes, elas podem possuir mais de uma dimensão.

Matrizes podem ser usadas para programar um jogo de xadrez ou damas, ou até mesmo armazenar as coordenadas (x,y) de um gráfico ou da posição de estrelas.

9.1 SINTAXE

A sintaxe para declaração de uma matriz é semelhante a declaração dos vetores. Mas considera-se também a quantidade de elementos das outras dimensões:

```
1 tipo nomeDaMatriz2D [NumeroLinhas][NumeroColunas];
2 tipo nomeDaMatriz3D [NumeroLinhas][NumeroColunas][NumeroProfundidades];
```

Por exemplo:

```
int MAT[3][4];
//matriz 3 linhas e 4 colunas do tipo inteiro
//esta matriz tem 12 elementos, com ndices de 0 a 2 para linhas e 0 a 3 para colunas
```

9.2 ACESSANDO MATRIZES

De forma similar aos vetores, as matrizes podem ter seus índices acessados individualmente

```
nomeDaMatriz [indice1][indice2]
```

Veja como exemplo o programa a seguir, o qual inicializa os elementos de uma matriz m com os valores iguais a soma dos índices de cada elemento e imprime cada valor.

```
#include <stdio.h>
int main()
{
    int m[3][2], i,j;
    for (i=0; i < 3; i++) {
        for(j=0; j < 2; j++) {
            m[i][j] = i+j;
            printf("i=%d j=%d elemento=%d\n", i, j, m[i][j]);
        }
    }
    return 0;
}
```

i=0	j=0	elemento=0
i=0	j=1	elemento=1
i=1	j=0	elemento=1
i=1	j=1	elemento=2
i=2	j=0	elemento=2
i=2	j=1	elemento=3

- O primeiro elemento da matriz assim como no vetor sempre recebe o índice "0"(zero).
- Em um vetor com N elementos, os índices portanto são numerados de "0"a "N-1". Logo, extendendo essa lógica para matrizes, em uma matriz com N por N elementos, os índices de cada linha/coluna são numerados de "0"a "n-1"também. Não é possível fazer uma operação com todo o conjunto dos dados armazenados na matriz de uma só vez!

Matrizes também são passadas para funções da mesma forma como os vetores são passados, com um pequeno detalhe: apenas a primeira dimensão pode ser omitida.

```
void imprimeMatriz(float m[3][3][4])
void imprimeMatriz(float m[][3][4])
```

Exercícios:

1. Faça um programa que leia uma matriz 5 x 5 e imprima a terceira coluna.
2. Faça uma função para calcular a multiplicação de uma matriz 3 x 4 por um escalar. Faça também uma função capaz de imprimir esta matriz.
3. **Desafio** - Crie um jogo da velha utilizando matrizes

Referências

Material - Turma X. Disponível: aqui.

CPlusPlus. Disponível: aqui.