Linguagem de Programação

Marcos Monteiro

MBA, ITIL, Perito Computacional Forense

Agenda

- Conceitos de lógica
- Conceitos de algoritmo
- Representações de algoritmo
- Algoritmo x Linguagem

Conceitos de lógica

- Lógica é uma ciência de índole **matemática** e fortemente ligada à **filosofia**.

- A aprendizagem da lógica **não constitui um fim em si mesmo**.

- Ela só tem sentido enquanto meio de garantir que nosso **pensamento proceda corretamente** a fim de chegar à **conhecimentos verdadeiros**.

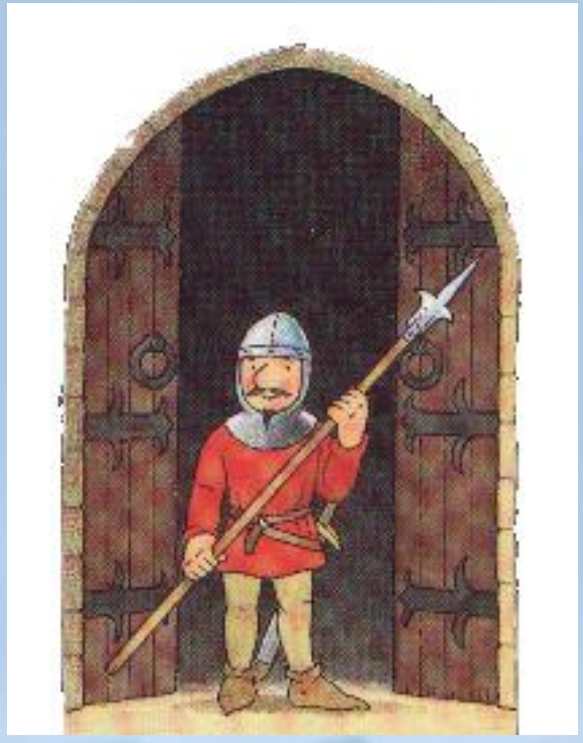
Conceitos de lógica

- A lógica é **extensivamente** usada em áreas como Inteligência Artificial e Ciência da computação.

- A programação lógica é uma tentativa de fazer computadores usarem **raciocínio lógico**, e a **linguagem de programação Java** é comumente utilizada para isto.

Teste de lógica

- Uma porta leva à morte; e outra, à liberdade.
- Um guarda é mentiroso e o outro não.
- **Só 1 Pergunta!!!**



Que pergunta você faria?

Conceitos de algoritmo

- Um algoritmo é uma **seqüência** não ambígua de instruções que é executada até que determinada condição se verifique.

- Eles podem **repetir passos** (fazer iterações) ou necessitar de **decisões** (tais como comparações ou lógica) até que a tarefa seja completada.

Conceitos de algoritmo

- Um algoritmo **não** representa necessariamente um **programa de computador**, e sim os **passos** necessários para realizar uma tarefa.

- Sua **implementação** pode ser feita por um **computador**, por outro tipo de máquina ou mesmo por um ser **humano**.

Conceitos de algoritmo

- Algoritmo para se vestir (versão 1.0):

- Algoritmo para se vestir (versão 2.0):

Conceitos de algoritmo

- Classificação por **implementação**
  - **Recursivo**
    - Invoca a si mesmo.
  - **Iterativo**
    - Usam estruturas de repetição.
  - **Determinístico**
    - Decisão exata.
  - **Não-Determinístico**
    - Deduz os melhores passos.

Conceitos de algoritmo

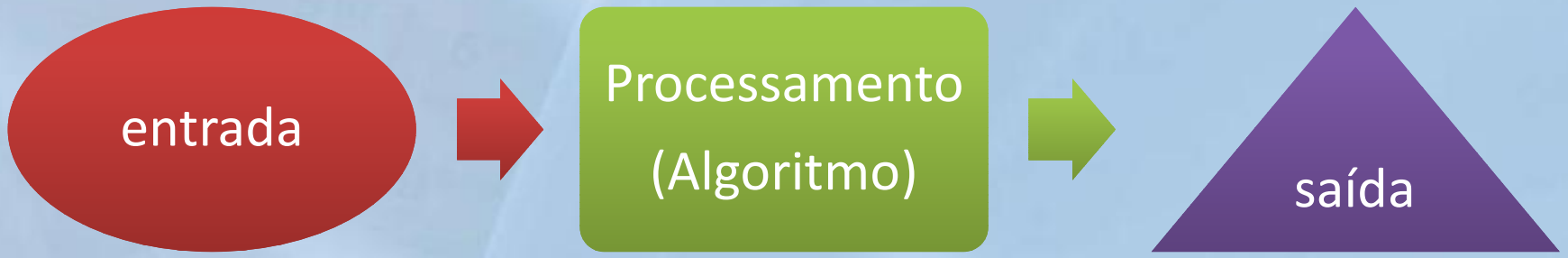
- Classificação por **implementação**
  - **Serial**
    - executados **instrução à instrução** individualmente, como uma lista de execução.
  - **Paralelo**
    - executados paralelamente, levando em conta arquiteturas de computadores com mais de um processador para executar mais de uma instrução ao mesmo tempo.

Conceitos de algoritmo

- Classificação por **paradigma**
  - **Divisão e conquista**
    - reduzem repetidamente o problema a sub-problemas. Geralmente, utilizando-se a forma recursiva, o sub-problema pode ser reduzido até que se torne pequeno o suficiente para ser resolvido.
  - **Paradigma heurístico e probabilístico**
    - algoritmos probabilísticos realizam escolhas aleatoriamente.
  - Existem outros...

Conceitos de algoritmo

- Execução de um algoritmo



  - **Entrada de dados**: é o ato de **fornecer dados** ao computador, que os manipulará de acordo com o que foi elaborado no algoritmo.
  - **Processamento**: consiste na manipulação dos dados armazenados de forma que **gere os resultados desejados**.
  - **Saída de dados**: é a ação, ou processamento, que o computador realiza para disponibilizar ao usuário os **resultados obtidos no processamento**.

Representações de algoritmos

- As formas mais comuns de representação de algoritmo são:
  - **Linguagem natural;**
  - **Fluxograma;**
  - **Pseudo-linguagem.**

Representações de algoritmos

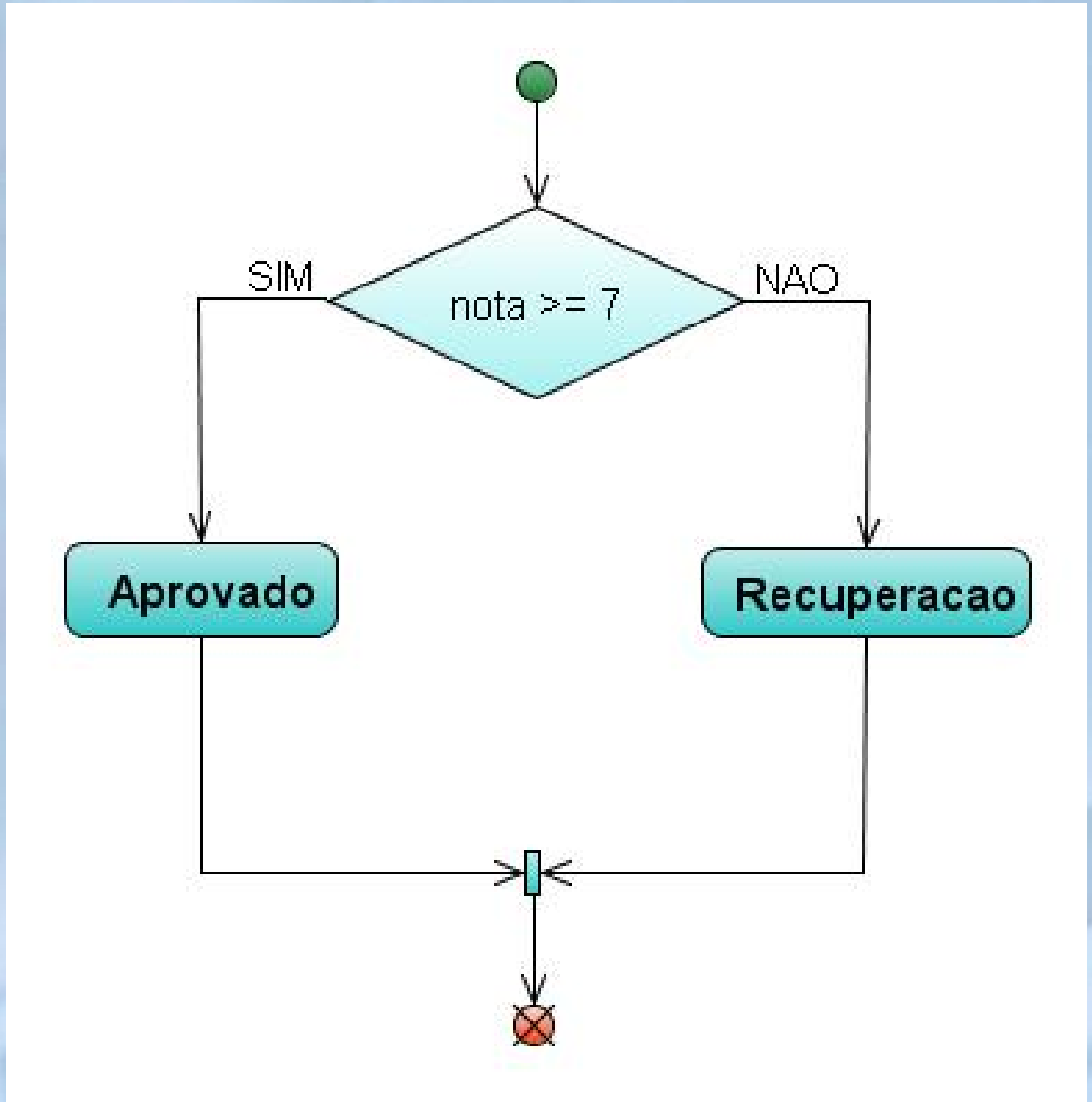
- Linguagem natural
  - Os algoritmos são expressos diretamente em linguagem natural.

Receita de bolo: Bata na batedeira, as claras em neve bem firme. Junte as gemas, uma a uma, e acrescente o açúcar. Despeje o leite aos poucos, sem parar de bater. Incorpore, por fim, delicadamente a farinha peneirada com o Chocolate em Pó e o fermento. Despeje em uma fôrma redonda (28 cm de diâmetro) untada e enfarinhada e leve para assar em forno quente (200º C) por aproximadamente 40 minutos...

Representações de algoritmos

- Fluxograma
  - Esta é um representação gráfica que emprega **formas geométricas padronizadas** para indicar as diversas **ações** e **decisões** que devem ser executadas para resolver o problema.

Representações de algoritmos



Representações de algoritmos

- Pseudo-linguagem

```
Algoritmo "SomaDeDoisValores"
declare
SOMA,A,B: inteiro
 inicio
   escreva("Digite dois números")
   Leia(A,B)
   SOMA <- A + B
   escreva(SOMA)
 fim
```

Algoritmo x Linguagem

Alto nível	Baixo nível
1 – Desce do carro	1 – Estacionar o carro no acostamento; 2 – Desligar o carro; 3 – Ligar a pisca alerta; 4 – Retirar o cinto de segurança; 5 – Abrir a porta; 6 – Colocar as pernas para fora do carro; 7 – Sair do veículo;
2 - Pega o estepe	8 – Abrir o porta malas; 9 – Pegar o triângulo de sinalização; 10 – Montar o triângulo; 11 – Colocar o triângulo no asfalto para fazer a sinalização; 12 – Retirar o macaco do porta malas; 13 – Colocar o macaco ao lado do carro; 14 – Retirar o estepe do porta malas; 15 – Colocar o estepe ao lado do carro;

Algoritmo x Linguagem

Alto nível	Baixo nível
3 – Troca o Pneu	16 – Colocar o macaco sob o carro; 17 – Girar a manivela para levantar o carro; 18 – Pegar a chave; 19 – Retirar os parafusos; 20 – Retirar o pneu; 21 – Colocar o estepe; 22 – Parafusar o pneu; 23 – Girar a manivela do macaco ao contrario; 24 – Colocar o pneu no porta malas; 25 – Colocar o macaco no porta malas; 26 – Guardar o triângulo; 27 – Fechar o porta malas;
4 – Volta a viajar	28 – Abrir a porta do carro; 29 – Sentar no banco; 30 – Colocar as pernas para dentro; 31 – Colocar o cinto de segurança; 32 – Ligar o carro; 33 – Continuar a viagem;

Agenda

- Tipos primitivos de dados
- Constantes e variáveis
- Expressões aritméticas
- Expressões lógicas

Estruturas de dados

- Estruturas de dados são caracterizadas pela **forma** como os dados são **organizados** (dispostos).

- Estruturas de dados ≠ Dados

  – Estruturas de dados **organizam** e **administram** os dados.

Estruturas de dados

- Estruturas de dados clássicas:
  - **Vetor ou Array**: linear e **estática**.
  - **Lista**: linear e **dinâmica**.
  - **Pilha**: baseada no princípio **LIFO** (*Last In*, *First Out*).
  - **Fila**: baseada no princípio **FIFO** (*First In*, *First Out*).
  - **Árvore**: possui um elemento **raiz**, elementos do tipo **nó** e elementos do tipo **folha**.
  - **Tabela**: estrutura de dados especial, que associa **chaves** de pesquisa (*hash*) a **valores**.

Tipos primitivos de dados

- Um tipo primitivo (também conhecido por nativo ou básico) é fornecido por uma linguagem de programação como um **bloco de construção básico**.
  - Numéricos
    - 1 -10 2008 3,14 -0,001
  - Caracteres (Alfanuméricos)
    - ‘C’, ‘#’, ‘a’, “Em JAVA isso não é tipo primitivo”
  - Booleano
    - Verdadeiro ou falso (*true*, *false*)

Tipos Primitivos

Constantes e Variáveis

- **Constantes**: representam valores fixos que não variam durante a execução do algoritmo.
  - Ex.: PI = 3,14

- **Variáveis**: representam valores que variam durante a execução do algoritmo.
  - Ex.: saldoContaCorrente = 1.350,77

Declaração e atribuição de variáveis

- Declaração é a **criação de locais na memória rotulados** com o identificador da variável (ou constante). Esse identificador será utilizado no algoritmo para a manipulação de um determinado tipo de informação.

- Atribuição é **dar um valor** a uma variável ou constante. Também usa-se o nome **inicialização** para indicar a primeira atribuição na variável ou constante.

Declaração e atribuição de variáveis

- **Declarando** variáveis:
  - **<tipo> <variável>;**

```
int a;
```

- **Declarando** e **inicializando** Variáveis:
  - **<tipo> <variável> = <expressão>;**

```
int a = 10;
```

- **Atribuindo** valores a variáveis:
  - **<variável> = <expressão>;**

```
a = 10;
```

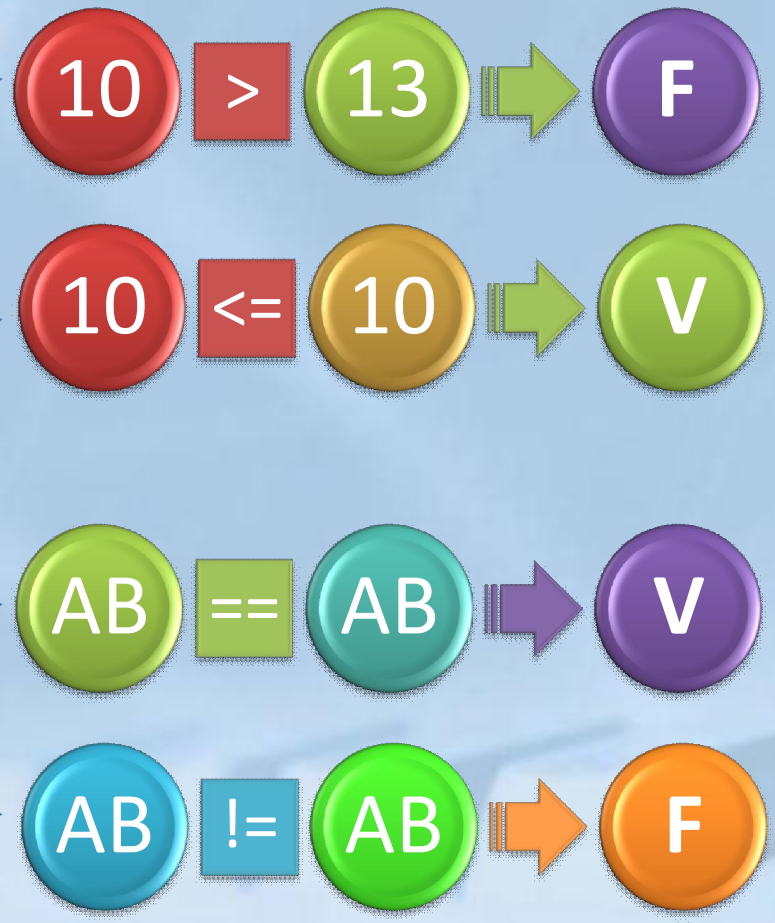
Operadores aritméticos

- **+** (Adição)
- **-** (Subtração)
- ***** (Multiplicação)
- **/** (Divisão)
- **%** (Modulo – Resto da divisão)

Exemplo da operação Módulo...

Operadores relacionais

- >
- >=
- <
- <=
- ==
- !=



Valores Booleanos (**Verdadeiro** ou **Falso**)

Operadores lógicos

- NOT (**!**)
- AND (**&** ou **&&**)
- OR (**|** ou **||**)

Operadores lógicos

Executar ou Compilar.

Existem dois métodos gerais pelos quais um programa pode ser executado.

•**Interpretador** -> Lê o código-fonte de seu programa uma linha por vez, executando a instrução específica nessa linha.

•**Compilador** -> Lê o programa inteiro e converte-o em um código objeto (código de máquina ou código binário).

Em geral, qualquer aplicativo no qual o programador digita seu programa interpreta e compila o programa.

- Comentários
- São linhas que você acrescenta em seu programa e que não serão interpretadas nem compiladas. Servem como uma informação sobre o que o programa faz naquela linha,naquele bloco,função etc.. Em C as linhas que serão comentários ficam entre /* *COMENTÁRIO* */. Ou antes da frase coloca-se duas barras //*COMENTARIO.*
- Exemplo:

```
#include <iostream>

        /* Um Primeiro Programa */
        int main ()
        {
            printf ("Ola! Eu estou vivo!\n");
            system("PAUSE > null");

        }
```

comentário

Como Construir um algoritmo

- **1 Análise Preliminar**
  - Entenda o problema, identifique os dados e os resultados.
- **2 Solução**
  - Desenvolver algoritmo para resolver o problema
- **3 Teste de qualidade (Computador Japonês/Teste de mesa)**
  - Ideal testar todas as combinações possíveis
- **4 Alteração**
  - Resultado do algoritmo não satisfatório, altere-o e submeta a um novo teste.
- **5 Produto Final**
  - Algoritmo concluído e testado.

Portugol

- Linguagem de algoritmo semelhante ao nosso português usada para desenvolver algoritmos (pseudocódigo).
- A representação de um algoritmo na forma de pseudocódigo é a seguinte:

Algoritmo *Nome_Do_Algoritmo*
variáveis
    Declaração das variáveis
Procedimentos
    Declaração dos procedimentos
Funções
    Declaração das funções
    Início
        Corpo do Algoritmo
    Fim

Exemplo 01.

```
Algoritmo Média
Variáveis
    real N1, N2, Média    //declaração das variáveis
Início
    Ler N1, N2            //atribui valores para N1 e N2
    Média => (N1+N2)/2    //cálculo e a variavel Média recebe o resultado
    Se Média >= 7 Então   // comando Se condicional
        Escrever "Aprovado"
    Senão
        Escrever "Reprovado"
    Fim de Se
Fim do programa
```

Exemplo 02.

Algoritmo *Area do trapézio*

Variáveis

real *bmaior,bmenor,altura,area*

Início

Ler *bmaior,bmenor,altura*

*area => (bmaior+bmenor)*altura/2*

escrever("A área do trapézio é", area)

Fim do programa

Portugol Adaptado para Linguagem C

- **Linguagens Estruturadas:**

Possuem três estruturas de programação básicas

1) **Estruturas de Seqüência** -> Um comando é executado logo após o outro, na ordem em que aparecem.

2) **Estruturas de Seleção** (comandos condicionais) -> Os comandos que estão dentro destas estruturas são executados SOMENTE se a condição de seleção for verdadeira.

3) **Estruturas de Iteração** (comandos de repetição) Os comandos que estão dentro destas estruturas são executados repetidas vezes, desde que a condição de repetição continue sendo verdadeira.

- **Comandos Básicos do Portugol:**

leia(*variável***);**
escreva(*variável***);**
variável ← *valor*; **// seta representa comando de atribuição**

Comandos das Estruturas de Seleção

Estrutura de seleção simples (se...)

se (*condição***)**
{
comando 1;
comando 2;
comando ...;
comando n;
}

Estrutura de seleção composta (se... senão...)

```
se (condição_a)
{
    comando 1_a;
    comando 2_a;
    comando ...;
    comando n_a;
}
senão
{
    comando 1_b;
    comando 2_b;
    comando ...;
    comando n_b;
}
```

- **Estrutura de seleção de escolha: (escolha... caso...)**

escolha (*variável***)**
{
    caso *valor1: comando 1;*
    caso *valor2: comando 2;*
    caso *valor3: comando 3;*
    caso *valor4: comando 4;*
caso contrário: *comando n;*

}

Comandos das Estruturas de Iteração:

- ***Repetição com condição no fim: (faça... enquanto...)***

faça
{
comando 1;
comando 2;
comando ...;
comando n;
} enquanto (*condição***)**

- ***Repetição com condição no início: (enquanto...)***

enquanto (*condição***)**
{
comando 1;
comando 2;
comando ...;
comando n;
}

Repetição com variável de controle: (para...)

para (*inicialização;condição;comandos de incremento***)**

{

comando 1;

comando 2;

comando ...;

comando n;

}

```
// Programa Exemplo em Portugol

/* Em Portugol não há necessidade
   de incluir as bibliotecas de
   comandos, como acontece na
   Linguagem C
*/


Programa Principal()
{
   // Declaração de variáveis
   inteiro horas;
   real valor,total;
   caracter resposta;

   //----> Início da iteração

   faça
   {
      // Entrada de dados
      escreva("Digite o nr. de horas:");
      leia(horas);
      escreva("Digite valor por hora:");
      leia(valor);

      // Processamento / Armazenamento
      total ← horas * valor;

      // Saída
      escreva("Total = ", total);

      escreva("Quer repetir? (S/N):");
      leia(resposta);
   } enquanto(resposta=='S')

   //----> Fim da iteração



   // Em Portugol não é necessário
   // incluir nenhum comando de parada
   // ao final


}
```

```
// Programa Exemplo em C

/* Inclusão de bibliotecas de
   comandos é obrigatória */

#include <stdio.h>
#include <conio.h>


main()
{
   // Declaração de variáveis
   int horas;
   float valor,total;
   char resposta;

   //----> Início da iteração

   faça
   {
      // Entrada de dados
      printf("Digite o nr. de horas:");
      scanf("%i",&horas);
      printf("Digite valor por hora:");
      scanf("%f",&valor);

      // Processamento / Armazenamento
      total = horas * valor;

      // Saída
      printf("Total = %f", total);

      printf("Quer repetir? (S/N):");
      resposta = getche();
   } while(resposta=='S')

   //----> Fim da iteração

   // Em C, costuma-se incluir um comando
   // de parada para que o usuário possa
   // visualizar o resultado antes que a
   // janela seja fechada

   getch();


}
```

Exemplo 01 (Portugol para C, média de 3 notas).

```
Programa principal()
{
real nota1,nota2,nota3,media;    // declaração de variáveis
{
escreva("Digite as tres notas");
leia(nota1);
leia(nota2);
leia(nota3);
Media     (nota1+nota2+nota3)/3;

Escreva ("A média é ", media);

}

}
```

MÓDULO I
Uma Visão Geral de C

- Inventada por Dennis Ritchie – Sist. Unix 1970
- Surgiu a partir da BCPL chamada de B.
- Segue o padrão ANSI (American National Standarts Institute).
- Linguagem de médio nível.
  - **Nível alto** –> *Pascal,Cobol,Fortran,Basic*.
  - **Nível Médio** –> *C++ C, Forth*.
  - **Nível baixo** –> *Assembler* .
- O C permite a manipulação de bits, bytes e endereços. Garantindo a *portabilidade*.
- Suporta o conceito de *tipo de dados*.

- Linguagem “quase estruturada”(a linguagem estruturada permite funções dentro de funções). Mais o C permite a *compartimentação* do código.
- O programa é todo sub-dividido em funções permitindo *modularização*(públicas e privadas).Figura.
- C é uma linguagem para programadores. Basic surgiu para facilitar a vida dos programadores. Assembler é uma linguagem de máquina.

```
if (x<10) {
          print ("Você digitou um número maior que 10");

          }
```

Forma de um Programa em C.

- *Case-sensitive*, else é diferente de ELSE.
- Todo programa é constituído de funções sendo a primeira a *main*.
- Bibliotecas . Executar operações de E/S, gráficas etc...

Compilar um programa em C

- *Criar* o programa.
- *Compilar* o programa.
- *Linkeditar* o programa com funções necessárias as bibliotecas.

C Versus C++

- C é a linguagem de origem, a base de tudo.
- C++ é a base estendida do C, projetada para suportar *OOP(Object Oriented Programming).*
- C++ aceita toda a linguagem C.
- Um compilador C++ funcionará com programacs C. *Compiladores C/C++.*

MÓDULO II
Expressões em C

- C suporta 5 tipos básicos de dados. *char,int,float,double e void.*
- char -> Caracteres ASCII.
- Int -> números inteiros.
- float -> ponto flutuante.
- double -> ponto flutuante.
- void ->declara função que não retorna valor ou cria ponteiros genéricos.
- Exceto o *void* os tipos básicos podem ter vários modificadores. São eles
  - signed,unsigned,long,short.

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis digitos de precisão
double	8	Dez digitos de precisão
long double	10	Dez digitos de precisão

Identificadores

Correto:	Incorreto:
contador	1contador
test23	cont!um
var1	var...3

- Case Sensitive.
- Evite nomes longos.
- um identificador não pode ser igual a uma palavra chave de C.

Palavras chaves em C (padrão ANSI)

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

Variáveis

- Posição nomeada de memória.
- Devem ser declaradas *antes* de serem usadas.
- Tipo de variável = tipo de dado.
- *tipo de dado_indentificador* ; (dessa forma que uma variável deve ser declarada) ex:
- São declaradas em 3 lugares

básicos: dentro das funções,na definição

dos parâmetros das funções,e fora de todas as funções.

- O nome da variável não tem nada a ver com seu tipo.

```
int valor1;
float valor2;
char frase;
```

Variáveis Locais ou Automáticas.

- Variáveis que são declaradas dentro de uma função são chamadas de variáveis locais ou automáticas.
- Não são reconhecidas fora de seu próprio bloco de código.
- A maioria dos programadores declara suas variáveis usadas por uma função imediatamente após o abre-chaves da função e antes de qualquer outro comando.
- Quando uma função é chamada, suas variáveis locais são criadas e, ao retornar, elas são destruídas.
- Parâmetros Formais -> São variáveis declaradas logo depois do nome da função e dentro dos parênteses (argumentos).

```
void func1(void)
{
    int x;

    x = 10;
}

void func2(void)
{
    int x;

    x = -199;
}
```

Variáveis Globais

- São reconhecidas pelo programa inteiro e podem ser usadas por qualquer pedaço de código.
- Você cria variáveis globais declarando-as fora de qualquer função.
- Podem ser acessadas por qualquer expressão independente de qual bloco de código contém a expressão.
- As variáveis globais são usadas qdo o mesmo dado é usado em muitas funções em seu programa.

```
#include <stdio.h>
int count;  /* count é global */

void func1(void);
void func2(void);

void main(void)
{
    count = 100;
    func1();
}

void func1(void)
{
   int temp;

   temp = count;
   func2();
   printf("count é %d", count); /* imprimirá 100 */
}

void func2(void)
{
   int count;
   for (count=1; count<10; count++)
     putchar('.');
}
```

Inicialização de Variáveis

- A forma geral de uma inicialização é:

  tipo nome_da_variável = constante;

- A atribuição de seu valor pode ser feita

no momento da sua criação ou após.

- Variáveis globais são inicializadas apenas no começo do programa.
- Variáveis locais são inicializadas cada vez que o bloco no qual são declaradas for inserido.

Alguns exemplos são

```
char ch = 'a';

int first = 0;

float balance = 123.23;
```

Constantes

- Referem-se a valores fixos que o programa não pode alterar.
- Podem ser de qualquer um dos cinco tipos de dados.
- Constantes caractere são envolvidas por aspas simples (‘a’), inteiras como já sabemos suportam apenas a classe dos inteiros, as de ponto flutuante podem ser float ou double.
- Constantes strings são um conjunto de caracteres colocados entre duas aspas (“isso é um teste”).
- String-> conjunto de caracteres. Caractere -> apenas um símbolo.
- Constantes Caractere de barra invertida ->Úteis para caracteres que não são facilmente inseridos através do teclado ou de strings (como por exemplo, o retorno de carro). Estes códigos são mostrados na tabela a seguir:

Código	Significado
\b	Retrocesso (BS)
\f	Alimentação de formulário (FF)
\n	Nova linha (LF)
\r	Retorno de carro (CR)
\t	Tabulação horizontal (HT)
\"	Aspas duplas
\'	Aspas simples
\0	Nulo
\\	Barra invertida
\v	Tabulação vertical
\a	Alerta (beep)
\N	Constante octal (onde N é uma constante octal)
\xN	Constante hexadecimal (onde N é uma constante hexadecimal)

Exemplo:

```
#include <stdio.h>
void main(void)
{
  printf("\n\tIsso é um teste");
}
```

Operadores

- C define quatro classes de operadores: aritméticos,relacionais,lógicos e bit a bit.
- Atribuição→ nome_da_variável= expressão;
- Conversão de tipos →refere-se à situação em que varíáveis de um tipo são misturadas com variáveis de outro. O valor do lado direito de uma atribuição é convertido no tipo do lado esquerdo(variável de destino:
- Operadores aritméticos:

Operador	Ação
-	Subtração, também menos unário
+	Adição
*	Multiplicação
/	Divisão
%	Módulo da divisão (resto)
--	Decremento
++	Incremento

Operadores Relacionais

Operadores relacionais

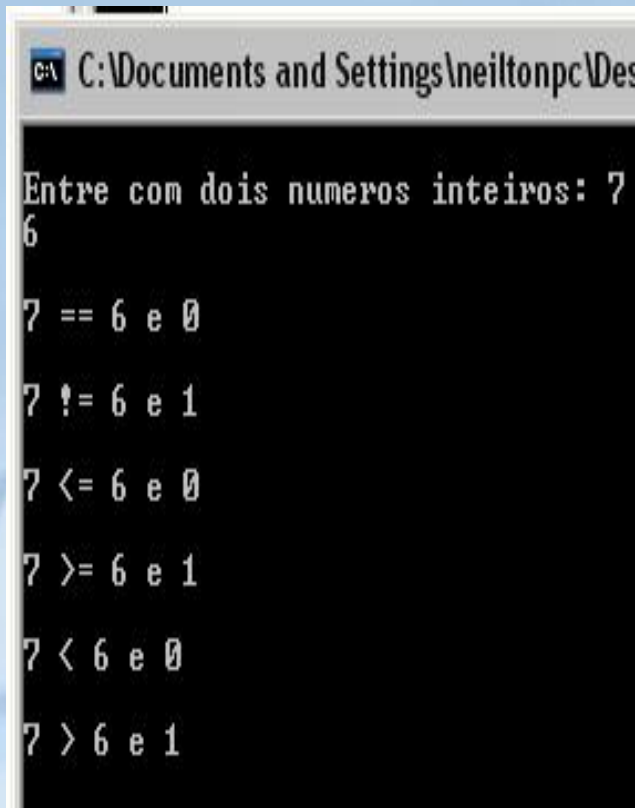
Operador	Ação
>	Maior que
>=	Maior que ou igual
<	Menor que
<=	Menor que ou igual
==	Igual
!=	Diferente

Operadores lógicos

Operador	Ação
&&	AND
! !	OR
!	NOT

Os operadores relacionais retornam verdadeiro (1) ou falso (0). Para verificar o funcionamento dos operadores relacionais, observe o programa abaixo:

```
int main()
{
 int i, j;
 printf("\nEntre com dois numeros inteiros: ");
 scanf("%d%d", &i, &j);
 printf("\n%d == %d é %d\n", i, j, i==j);
 printf("\n%d != %d é %d\n", i, j, i!=j);
 printf("\n%d <= %d é %d\n", i, j, i<=j);
 printf("\n%d >= %d é %d\n", i, j, i>=j);
 printf("\n%d < %d é %d\n", i, j, i<j);
 printf("\n%d > %d é %d\n", i, j, i>j);


 }
```



Introdução as Entradas e Saídas →printf() e scanf()

- Função printf()

  Protótipo para printf():

int printf (const char **string_de_controle*,...);

- A string de controle consiste em dois tipos de itens. O primeiro tipo é formado por caracteres q serão impressos na tela, o segundo contem comandos de formato q definem a maneira pela qual os argumentos subsequentes serão mostrados.
- Os comandos de formato começam

com (%)e seguido pelo código do formato.

Observe a tabela.

Código	Formato
%c	Caractere
%d	Inteiros decimais com sinal
%i	Inteiros decimais com sinal
%e	Notação científica (e minúsculo)
%E	Notação científica (E maiúsculo)
%f	Ponto flutuante decimal
%g	Usa %e ou %f, o que for mais curto
%G	Usa %E ou %F, o que for mais curto
%o	Octal sem sinal
%s	String de caracteres
%u	Inteiros decimais sem sinal
%x	Hexadecimal sem sinal (letras minúsculas)
%X	Hexadecimal sem sinal (letras maiúsculas)
%p	Apresenta um ponteiro
%n	O argumento associado é um ponteiro para inteiro no qual o número de caracteres escritos até esse ponto é colocado
%%	Escreve o símbolo %

- Vamos ver alguns exemplos de printf() e o que eles exibem:
- printf ("Teste %% %%") -> "Teste % %"
- printf ("%f",40.345) -> "40.345"
- printf ("Um caractere %c e um inteiro %d",'D',120) -> "Um caractere D e um inteiro 120"
- printf ("%s e um exemplo","Este") -> "Este e um exemplo"
- printf ("%s%d%%","Juros de ",10) -> "Juros de 10%"

- A função scanf()

O protótipo para scanf() é :

Int scanf (const char *string_de_controle,...);

- A função scanf() devolve o número de items de items de dados que foi atribuido, com êxito a um valor.

- Os especificadores de formato de entrada são precedidos por um sinal % e informam a scanf() que tipo de dado deve ser lido imediatamente após.

Código	Significado
%c	Lê um único caractere
%d	Lê um inteiro decimal
%i	Lê um inteiro decimal
%e	Lê um número em ponto flutuante
%f	Lê um número em ponto flutuante
%g	Lê um número em ponto flutuante
%o	Lê um número octal
%s	Lê uma string
%x	Lê um número hexadecimal
%p	Lê um ponteiro
%n	Recebe um valor inteiro igual ao número de caracteres lidos até então
%u	Lê um inteiro sem sinal
%[]	Busca por um conjunto de caracteres.

Exemplos:

Inserindo números

```
#include <stdio.h>

void main(void)
{
   int i, j;

   scanf("%o%x", &i, &j);
   printf("%o %x", i, j);
}
```

Lendo caracteres individuais

```
scanf("%c%c%c", &a, &b, &c);
```

Lendo strings

```
#include <stdio.h>

void main(void)
{
  char str[80];

  printf("entre com uma string: ");
  scanf("%s", str);
  printf("eis sua string: %s", str);
}
```

Funções

- Funções são as estruturas que permitem ao usuário separar seus programas em blocos. Se não as tivéssemos, os programas teriam que ser curtos e de pequena complexidade. Para fazermos programas grandes e complexos temos de construí-los bloco a bloco.
- Uma função no C tem a seguinte forma geral:

tipo_de_retorno nome_da_função (declaração_de_parâmetros)
{
corpo_da_função
}

- O tipo-de-retorno é o tipo de variável que a função vai retornar. O default é o tipo **int**, ou seja, uma função para qual não declaramos o tipo de retorno é considerada como retornando um inteiro. A declaração de parâmetros é uma lista com a seguinte forma geral: *tipo nome1, tipo nome2, ... , tipo nomeN*
- Repare que o tipo deve ser especificado para cada uma das N variáveis de entrada. É na declaração de parâmetros que informamos ao compilador quais serão as entradas da função (assim como informamos a saída no tipo-de-retorno).
- O corpo da função é a sua alma. É nele que as entradas são processadas, saídas são geradas ou outras coisas são feitas.

- O comando **return** tem a seguinte forma geral:
- *return valor_de_retorno;* ou *return;*
- Digamos que uma função está sendo executada. Quando se chega a uma declaração **return** a função é encerrada imediatamente e, se o valor de retorno é informado, a função retorna este valor. É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.
- **Protótipos de Funções são nada mais, nada menos, que declarações de funções. Isto é, você declara uma função que irá usar. O compilador toma então conhecimento do formato daquela função antes de compilá-la. O código correto será então gerado. Um protótipo tem o seguinte formato:**

  tipo_de_retorno nome_da_função (declaração_de_parâmetros);

```c
#include <stdio.h>

float Square (float a);

int main ()

{

    float num;
    printf ("Entre com um numero: ");
    scanf ("%f",&num);
    num=Square(num);
    printf ("\n\nO seu quadrado vale: %f\n",num);
    return 0;
}

float Square (float a)

{

    return (a*a);
}
```

Observe que a função **Square()** está colocada depois de **main()**, mas o seu protótipo está antes. Sem isto este programa não funcionaria corretamente.

- **O Tipo void ele nos permite fazer funções que não retornam nada e funções que não têm parâmetros! Podemos agora escrever o protótipo de uma função que não retorna nada:**

  void nome_da_função (declaração_de_parâmetros); ou

  tipo_de_retorno nome_da_função (void); ou

  void nome_da_função (void);

```c
#include <stdio.h>

void Mensagem (void);

int main ()

{
     Mensagem();
     printf ("\tDiga de novo:\n");
     Mensagem();
     return 0;
}

void Mensagem (void)

{

     printf ("Ola! Eu estou vivo.\n");
}
```

Continuação – Funções
Chamada por Valor, Chamada por Referência

Chamada por Valor

Os programas passam informações para funções usando parâmetros. Quando um parâmetro é passado a uma função, a Linguagem C++ usa uma técnica conhecida como **chamada por valor** para fornecer à função uma cópia dos valores dos parâmetros. Usando a chamada por valor, quaisquer modificações que a função fizer nos parâmetros existem apenas dentro da própria função. Quando a função termina, o valor das variáveis que a função chamadora passou para a função não é modificada dentro da função chamadora.

O programa a seguir irá imprimir " 100 10". O valor do argumento quadrado() ,10 é copiado no parâmetro x. Qdo a atribuição x=x*x ocorre, apenas a variável local x é modificada. A variável t, usada para chamar quadrado(), ainda tem o valor 10.

```c
#include <iostream>

int quadrado(int x);

int main()
{
      int t=10;
      printf("%d  %d", quadrado(t),t);
      system("PAUSE>null");

}
int quadrado(int x)
{
         x=x*x;
         return(x);
}
```

Chamada por Referência

A diferença entre chamada por valor e chamada por referência é que, usando a chamada por valor, as funções recebem uma cópia do valor de um parâmetro. Por outro lado, com a chamada por referência, as funções recebem o endereço de memória da variável. Portanto, as funções podem alterar o valor armazenado na posição de memória específica (em outras palavras, o valor da variável); alterações essas que permanecem após a função terminar.

COMANDOS DE CONTROLE DO PROGRAMA

- O padrão ANSI divide os comandos de C nestes grupos:
- *Seleção*
- *Iteração (repetição)*
- *Desvio*
- *Rótulo*
- *Expressão*
- *Bloco*

Comandos de Seleção

- C suporta dois tipos de comandos de seleção: ***if e swith***.
- If → A forma geral da sentença if é :

If (expressão) comando;

else comando;

- O comando if pode ser alinhado com outros ifs.
- swith → Comado interno de seleção múltipla . Forma geral:

swith(expressão){
Case constante1:
Comandos
break;
case constante2:
comandos
break; }

```
#include <iostream>


        int main ()

        {
            float nota1,nota2,nota3,media;

            printf ("Digite as tres notas\n");
            scanf("%f%f%f",&nota1,&nota2,&nota3);
            media=(nota1+nota2+nota3)/3;
            printf("\nA media e %.2f",media);
            if (media>=7) printf ("\nO aluno esta aprovado");
            else
            if (media<7)printf ("\nO aluno esta de AF");
            system("PAUSE > null");
        }
```

Observe o programa acima, são digitadas 3 notas, se o aluno tirar media acima ou igual a 7 imprime "O aluno esta aprovado", senão se ele tirar abaixo de sete está de AF. Modifique-o para que qdo ele tirar abaixo de sete e acima ou igual a 4 está de AF, abaixo de 4 estará reprovado.

```
#include <iostream>

float calc_media(float nota1,float nota2,float nota3);
        int main ()

        {
            float a,b,c,media;
            printf ("Digite as tres notas\n");
            scanf("%f%f%f",&a,&b,&c);
            media=calc_media(a,b,c);
            printf("\nA media e %0.3f",media);
            if (media>=7) printf ("\nO aluno esta aprovado");
            else
            if (media<7)printf ("\nO aluno esta de AF");
            system("PAUSE > null");
        }

float calc_media(float nota1,float nota2,float nota3)
{
  return(nota1+nota2+nota3)/3;

}
```

Este programa é identico ao programa anterior, so que este utiliza uma função extra chamada calc_media para realizar o cálculo da média.

```
#include <iostream>

        int main ()

        {
            int nota1,nota2,nota3,soma;
            printf ("Digite as tres notas\n");
            scanf("%d%d%d",&nota1,&nota2,&nota3);
            soma=(nota1+nota2+nota3);
            printf("\nA soma e %d",soma);

            switch (soma)
            {
                    case 15: printf("\n O aluno tirou 15");break;
                    case 18: printf("\n O aluno tirou 18");break;
                    case 9: printf("\n O aluno tirou 9 se lascou");break;

                    }

            system("PAUSE > null");
        }
```

Este código soma tres notas, mostra sua soma, em seguida entra no comando switch . Caso a soma das três notas for 15, imprime "O aluno tirou 15", se tirar 18 e tirar 9 com suas frases correspondentes. Obs, o comando switch so trabalha com variáveis inteiras.

Comandos de Iteração

- Também chamados de Laços, permitem que um conjunto de instruções seja executado até que ocorra uma certa condição.
- Os principais laços são o *for, while e do-while*.
- A forma geral do comando for é:

for (inicialização; condição; incremento) comando;

```
#include <iostream>

int main ()
{
     int x;
     for (x=1; x<=100; x++)
     printf("%d ",x);
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
89 90 91 92 93 94 95 96 97 98 99 100 neilton@ubuntu
```

Este programa imprime os números de 1 a 100 na tela

O laço for pode conter múltiplos comandos

```
#include <iostream>

int main()
{
     int x,y;
     float z;
     for (x=100; x!=65; x-=5)
     {
       z=x*x;
       printf("O quadrado de %d, %.2f ",x,z);
     }
     system("PAUSE>null");
}
```

```
O quadrado de 100, 10000.00 O quadrado de 95, 9025.00 O
quadrado de 90, 8100.00 O quadrado de 85, 7225.00 O quad
rado de 80, 6400.00 O quadrado de 75, 5625.00 O quadrado
 de 70, 4900.00
```

Tanto a multiplicação de x por si mesmo como a função printf() são executadas até que x seja igual a 65. Note que o laço é executado de forma inversa: x inicia de 100 e é decrementado de 5 cada vez que o laço se repete.

O laço *while* :

- Sua forma geral é:

while (condição) comandos;

Exemplo:

```
#include<iostream>
int main()
{
    int x;
    x=0;
    while (x<10) {
        x=x+1;
        printf("Contagem %d\n",x);
        }
        system("PAUSE>null");
        }
```

```
Contagem 1
Contagem 2
Contagem 3
Contagem 4
Contagem 5
Contagem 6
Contagem 7
Contagem 8
Contagem 9
Contagem 10
```

Este programa mostra uma contagem de 0 ate 10. A variavel x inicialmente é zerada e entra no laço while, enquanto x for menor que 10 x é incrementada de 1 e mostra a contagem.

O laço *do-while*:

Sua forma geral é:

do {

comandos;

} while (condição);

Exemplo:

```
#include<iostream>
int main()
{
    int x,y;
    do {
        scanf("%d%d",&x,&y);
        printf("Soma %d\n",x+y);
        }while (x+y!=100);
        printf("Acabou");

        system("PAUSE>null");
        }
```

```
3
2
Soma 5
5
7
Soma 12
50
50
Soma 100
Acabou
```

Diferente do laço for e while, a condição é testada no fim do laço. Este exemplo, faz a soma de dois números digitados, enquanto a soma for diferente de 100 não se sai do laço, qdo a soma for 100 quebra-se o laço e mostra "Acabou".

Comandos de Desvio

Os principais comandos de desvio incondicional são *return,goto,break e continue*.

Sendo que o *return* e o *goto* vc pode usar em qualquer lugar do programa.E os comandos *break* e *continue* em conjunto com qualquer dos comandos de laço.

return → Usado para retornar de uma função.

return *expressão;*

goto → Pouco usado na linguagem C convencional, necessita de um rótulo para sua operação padrão.

goto rótulo

.

rótulo:

break → Usado apara terminar um *case* no laço *switch* ou em qualquer laço.

continue →tem sua forma um pouco parecida com o *break*,porém ao invés de forçar a terminação, faz q ocorra a próx. Interação do laco.

Ponteiros

Ponteiro é uma variável que contém um endereço de memória.Este endereço é normalmente a posição de outra variável na memória.
A forma geral de declarar uma variável ponteiro é:
*tipo *nome;*

Operadores de ponteiros:
Existem dois operadores especiais para ponteiros; * e &.

- O & é um operador unário que devolve o endereço na memória do seu operando. Ex:

m=&count; → coloca em m o o endereço da memória que contém a variável count. ("o endereço de").

- O * é o complemento de & , devolve o valor da variável localizada no endereço que o segue. Ex:

*q=*m;* →coloca em q o que está no endereço de m. ("no endereço").

```c
#include <iostream>

int main()
{
    int valor=124;
    int *endereco;
    endereco=&valor;
    printf("\nA variavel armazena o seguinte valor %d",valor);
    printf("\nO endereco da variavel e %d",endereco);
    system("PAUSE>null");

}
```

```
A variavel armazena o seguinte valor 124
O endere♦o da variavel e 6618820
```

Neste programa e criada uma variavel comum inteira(valor) e é logo atribuido o valor de 124 para ela. Tambem e criada uma ponteiro(endereco) para receber o endereço da variavel valor.Em seguida é mostrado o valor de ambas.

MATRIZES

Matrizes são variáveis que contém vários valores de um mesmo tipo. Por exemplo, podemos criar a matriz notas para armazenar as notas obtidas por 100 alunos em um exame, ou então utilizar uma matriz chamada gastos_mensais para anotar nossos gastos mensais ao longo do ano. Uma matriz armazena vários valores de um mesmo tipo: podemos criar matrizes para armazenar qualquer um dos tipos básicos de variáveis, como int, float e char. Cada valor é armazenado separadamente em um elemento da matriz, e pode ser acessado e modificado a qualquer momento.

Declaração de uma matriz

Para criar uma matriz, precisamos declarar três atributos dela:

* O tipo de valor que vai ser armazenado na matriz
* O nome da matriz, para que possamos acessá-la
* O número de elementos da matriz

A declaração de uma matriz é muito parecida com a declaração de uma variável, bastando adicionar o número de elementos que desejamos que ela tenha. A sintaxe é a seguinte:

<tipo> <nome> [<numero de elementos>];

Por exemplo, caso quiséssemos criar uma matriz chamada catálogo para armazenar 156 inteiros, a declaração seria assim:

```
int catalogo [156];
```

- **Podemos utilizar qualquer tipo de variáveis já estudadas anteriormente para criar uma matriz, como float, int, char.**

- **Uma vez criada uma matriz de um determinado tipo, ela só pode receber valores deste tipo.**

- **Assim como uma variável normal, podemos atribuir valores para uma matriz no momento de sua declaração. Isto é feito utilizando o operador de atribuição "=" seguido dos valores contidos entre chaves e separados por vírgulas.**

```
int teste[5] = { 1, 2, 3, 4 , 5};
```

Acessando Valores de uma Matriz

- Após criar uma matriz, podemos acessar qualquer valor dentro dela. Cada valor, ou elemento de uma matriz, possui um número próprio. Toda matriz começa no elemento 0.

- Precisamos ter isso em mente quando acessamos valores dentro de uma matriz, pois o primeiro elemento será o elemento “0”, o segundo elemento será o elemento “1”.

- Cada elemento de uma matriz é tratado como uma variável separada.

- Assim, podemos atribuir valor para um elemento, exibí-lo na tela, utilizá-lo em operações matemáticas e em laços condicionais. O programa abaixo ilustra estas várias ações:

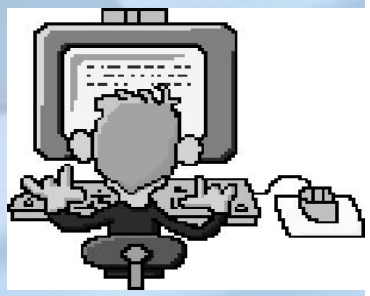
```c
#include<iostream>

int main()
{
    int matriz[5]={3,4,5,6,7};
    printf("\nO primeiro elemento da matriz e %d",matriz[0]);
    printf("\nO ultimo elemento da matriz e %d",matriz[4]);
    printf("\nA soma do segundo e o quarto elemento e %d",(matriz[1]+matriz[3]));
    matriz[4]=26;
    printf("\nMudamos o quinto elemento da matriz para %d",matriz[4]);
    system("PAUSE>null");
```

```
O primeiro elemento da matriz e 3
O ultimo elemento da matriz e 7
A soma do segundo e o quarto elemento e 10
Mudamos o quinto elemento da matriz para 26
```

Utilizando Laços para Percorrer Matrizes

- Podemos utilizar qualquer um dos laços que estudamos, mas sem dúvida o laço for é o mais prático para trabalhar-se com matrizes.

- Utilizamos a variável de controle do laço para acessar cada um dos elementos desejados (lembre-se que a matriz sempre começa no elemento 0), como vemos no programa abaixo que percorre os elementos de uma matriz, primeiro preenchendo a matriz com os dados entrados pelo usuário, depois exibindo estes dados na tela.



```c
int main()
{
int sequencia[4];
for (int i = 0; i < 4; i++) {
printf("Entre com o elemento numero %d da sequencia:",(i+1));
scanf("%d",&sequencia[i]);
}
printf("A sequencia entrada pelo usuario foi: ");
for (int i = 0; i < 4; i++) {
printf(" %d ",sequencia[i]);
}
system("PAUSE > null");

}
```

```
Entre com o elemento numero 1 da sequencia:3
Entre com o elemento numero 2 da sequencia:4
Entre com o elemento numero 3 da sequencia:5
Entre com o elemento numero 4 da sequencia:7
A sequencia entrada pelo usuario foi:  3  4  5  7
```

Importante: como vimos no exemplo anterior, podemos utilizar variáveis para acessar os elementos de uma matriz. Da mesma forma, podemos definir constantes para indicar o número de elementos de uma matriz. Essa técnica é muito útil, pois caso precisemos alterar o número de elementos da matriz ao invés de caçarmos no código todas as referências à este número, tudo que precisamos fazer é alterar o valor da constante.

```
#include <iostream>

int main()
{
const int TAMANHO = 4;
int sequencia[TAMANHO];
for (int i = 0; i < 4; i++) {
printf("Entre com o elemento numero %d da sequencia:",(i+1));
scanf("%d",&sequencia[i]);
}
printf("A sequencia entrada pelo usuario foi: ");
for (int i = 0; i < 4; i++) {
printf(" %d ",sequencia[i]);
}
system("PAUSE > null");

}
```

Matrizes Multidimensionais

Além das matrizes simples de uma única dimensão, C++ permite a criação de matrizes de múltiplas dimensões. As matrizes bidimensionais são sem dúvida as mais utilizadas e as mais úteis, pois comportam-se como tabelas com linhas e colunas. Ao declarar uma matriz multidimensional, adicionamos um conjunto de colchetes para cada dimensão extra. Entre os colchetes de cada dimensão, colocamos o número de elementos que aquela dimensão terá (ou uma variável que represente o número de elementos). Assim:

```
int tabela [10] [5]; //matriz bidimensional
int horas [12] [30] [24]; //matriz de três dimensões
int minutos [12] [30] [24] [60]; //matriz de quatro dimensões
```

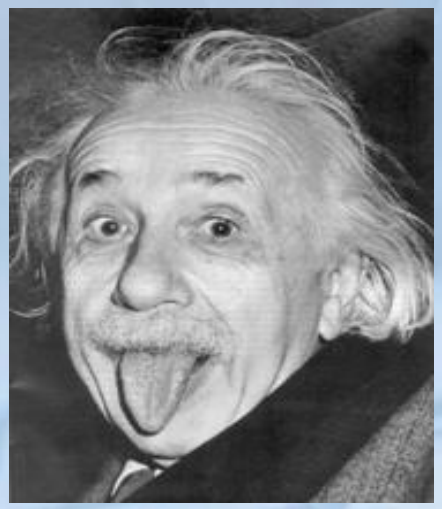
Normalmente trabalhamos no máximo com matrizes bidimensionais, mas podem surgir ocasiões onde matrizes de mais de duas dimensões sejam necessárias. Matrizes multidimensionais funcionam como matrizes dentro de matrizes. Por exemplo, uma matriz bidimensional pode ser vista como uma matriz de uma dimensão cujos elementos são outra matrizes. Esta analogia é útil para entender como é feita a inicialização dos valores de matrizes multidimensionais: inicializamos a matriz separando seus elementos por vírgulas, onde cada elemento é uma matriz individual e é inicializada da mesma forma que a matriz "principal". Por exemplo, seja a matriz bidimensional tabela:

```
int tabela [2] [3] = { {  9, 2, 3} , { 4, 5, 6}};
```

Exemplo de programa utilizando uma matriz tridimensional

```
int main()
{
    int i,j,k,tritabela[2][2][2]={{{9,8},{7,1}},{{4,3},{0,1}}};

    for (i=0;(i<2);i++)
    for (j=0;(j<2);j++)
    for (k=0;(k<2);k++)
    printf(" %d \n",tritabela[i][j][k]);



    system("PAUSE>null");
}
```

Analise o que este programa faz. Analise cada passo, só lembrando que este trabalha com uma matriz tridimensional.



Vai uma quadridimensional??