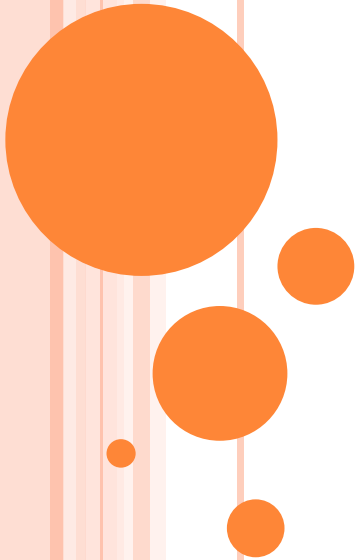


Paradigmas de Linguagens de Programação

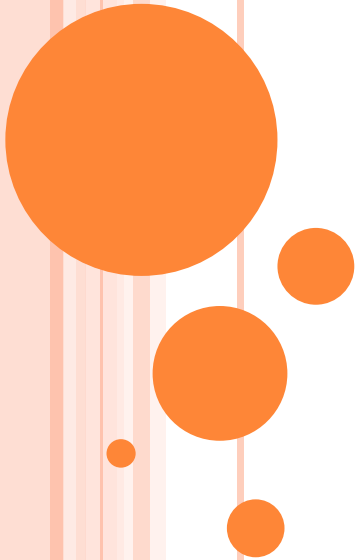
Aula 03

Prof. Adriana Bueno

adriana.bueno@ulbra.br

TÓPICOS

- Elementos que compõe um programa
- Modularização
- Funções



Elementos que compõe um programa

VARIÁVEIS

- Uma variável de programa é uma abstração de uma célula ou de um conjunto de células de memória do computador.

- As variáveis são utilizadas para guardar valores que poderão ser utilizados durante o processamento de um programa.

- Os seis atributos de uma variável são: nome, endereço, valor, tipo, tempo de vida e escopo.

VARIÁVEIS

- Os nomes de variáveis são os nomes mais comuns em programas. É o que identifica a variável dentro de um programa.
- O endereço de uma variável é o mesmo da memória à qual ela está associada.
- O tipo de uma variável determina a faixa de valores que ela pode ter e o conjunto de operações definidas para os valores do tipo.

VARIÁVEIS

- O valor de uma variável é o conteúdo da célula ou das células de memória associadas à variável.
- O escopo é o intervalo de linhas de programa que reconhece a variável e todos os seus atributos.
- O tempo de vida é o intervalo de tempo em que a variável está sendo usada e está ativa no programa.

VARIÁVEIS GLOBAIS E LOCAIS

- Globais: Variáveis visíveis em todo o programa.
- Locais: Variáveis declaradas em módulo e visíveis onde foram declaradas.

Tipos de dados

- Ao declararmos variáveis ou na utilização de valores em programas, é necessário definir o domínio ou faixa de valores que podem ser armazenados, ou seja, os tipos de dados que estas variáveis podem receber.

- Os tipos de dados podem ser primitivos ou derivados.

Primitivos

- São os tipos de dados nativos da linguagem.
- Tipos de dados que não são definidos em termos de outros tipos.
- Exemplos:
  - Tipo Numérico
  - Inteiro, Ponto Flutuante, Decimal, ..)
  - Tipo Booleano (valores verdade)
  - Tipo Caracter ou Cadeia de Caracteres.

Derivados ou Compostos

- São formados pela composição de dados primitivos. Isto é feito nas linguagens pela definição de registros. Por exemplo, na linguagem C podemos definir um registro da seguinte forma:

```
struct registro_cliente
{
  char nome[30];
  float salario;
  int idade;
};
```

Estruturas de controle

- São estruturas que permitem controlar **fluxo de execução do programa.** Normalmente utilizam condições lógicas que determinam o que deve ser feito em tempo de execução.

  - Se ... senao (if ... Else)
  - Faca_caso (switch case)
  - Para (for)
  - Enquanto (while)

EX:

Português

```
se (n=1) entao
        escreva( ‘numero um’ )
senao
        escreva ( ‘numero diferente de um’ );
```

C

```
if (n==1)
        printf( ‘numero um’ )
else
        printf ( ‘numero diferente de um’ );
```

EX:

Português

```
para i de 0 ate 9 passo 1 faca
        escreva (i)
fimpara
```

C

```
for (i=1; i<=10; i++)
 printf ("\n %d", i);
```

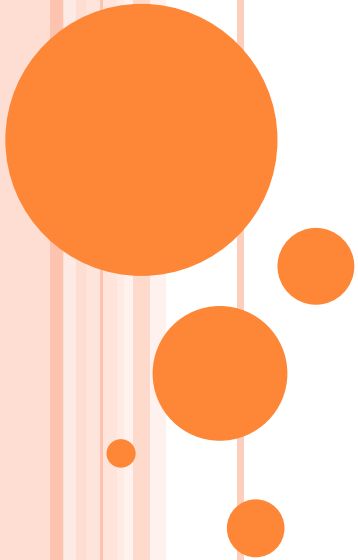
EX:

Português

```
enquanto (x>=0) faca
        soma:= soma+x
        leia (x)
fimenquanto
```

C

```
while (x>=0){
        soma = soma + x;
        scanf (“%d”, &x);
}
```



MODULARIZAÇÃO

MODULARIZAÇÃO

- Muitas vezes um problema grande pode ser resolvido mais facilmente se for dividido em pequenas partes.

- Tratar partes menores e mais simples em separado é muito mais fácil do que tratar o problema grande e difícil de uma vez.

- Se for necessário desenvolver e testar programas mais sofisticados (que envolvam mais funcionalidades) deveremos usar técnicas que nos permitam, de alguma forma, organizar o código fonte.

MODULARIZAÇÃO

- É comum em programação decompor programas complexos em programas menores e depois juntá-los para compor o programa final.

- Essa técnica de programação é denominada programação modular ou modularização de programas.

MODULARIZAÇÃO

- A programação modular facilita a construção de programas grandes e complexos, através de sua divisão em pequenos módulos, ou subprogramas, mais simples. Estes subprogramas além de serem mais simples de serem construídos, são também mais simples de serem testados.

- Esta técnica também possibilita o reaproveitamento de código, pois podemos utilizar um módulo quantas vezes for necessário eliminando a necessidade de escrevê-lo repetidamente.

MODULARIZAÇÃO

- Outro aspecto importante da modularização é a possibilidade de vários programadores trabalhem simultaneamente na solução de um mesmo problema, através da codificação separada dos módulos. Assim, cada equipe pode trabalhar em um certo conjunto de módulos ou subprogramas. Posteriormente estes módulos são integrados formando o programa final.

MODULARIZAÇÃO

- Em outras palavras, quando for necessário construir um programa grande, devemos dividi-lo em partes e então desenvolver e testar cada parte separadamente. Mais tarde, tais partes serão acopladas para formar o programa completo.

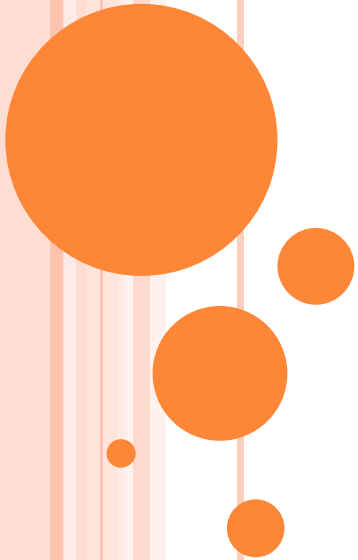
- Por exemplo, a modularização em C começa através do uso adequado de funções. Funções são algumas das formas usadas para agruparmos código de forma organizada e modularizada.

MODULARIZAÇÃO

- Quando um programa utiliza a modularização (divisão em módulos) dizemos que ele é não monolítico. Por outro lado, um programa monolítico é aquele que é escrito em um único bloco sequencial de código.

- Quando temos na modularização um bloco de código definido dentro de outro ocorre o que chamamos de bloco aninhado.

- **Registro de Ativação:** é a chamada para execução de um módulo.



FUNÇÕES

FUNÇÕES

- Funções são elementos fundamentais em toda linguagem de programação, já que são ferramentas essenciais para abstração em programação.

- Em linguagens diferentes, as funções são conhecidas variavelmente como procedimentos, sub-rotinas, subprogramas ou métodos, e possuem diversas características em comum, assim como algumas diferenças importantes nas mais variadas linguagens.

TERMINOLOGIA BÁSICA

- Fortran e Ada fazem distinção entre funções que retornam um valor e aquelas que não retornam. As primeiras são denominadas *Funções*, e as últimas, *Sub-rotinas* em Fortran e *Procedimentos* em Ada.

- Linguagens do tipo de C não fazem tal distinção; ambas são chamadas de *Funções*.

- Em C++ e Java, um *método* é uma função declarada dentro de uma classe.

CHAMADA E RETORNO DE FUNÇÕES

- O controle retorna de uma função chamada de uma entre duas formas:
  - Para uma função ou um procedimento *void*, o controle retorna quando o final do corpo da função é atingido ou quando uma declaração *return* é encontrada.
  - Para uma função *não-void*, o controle retorna quando uma declaração *return* é encontrada. Neste caso, a declaração *return* tem a expressão que designa o valor a ser retornado.

Exemplo em C/C++

```
int h,i;

void B (int w){
        int j,k;
        i = 2*w;
        w = w+1;
}

void A (int x, int y) {
        bool i,j;
        B(h);
}

int main (){
        int a, b;
        h = 5; a = 3; b = 2;
        A(a,b);
}
```

EXEMPLO

- No programa exemplo, **main** é inicialmente chamado e as variáveis globais **h** e **i** estão disponíveis para ele, junto às variáveis locais **a** e **b**.

- Após as três primeiras declarações de atribuições em **main** terem sido executadas, as variáveis **h**, **a** e **b** têm valor 5, 3 e 2, respectivamente.

- Quando **A** é chamada por **main**, seus parâmetros **x** e **y** obtêm os valores 3 e 2, e **A** também tem acesso à variável global **h**.

EXEMPLO

- As variáveis locais de **A**, **i** e **j** também são acessíveis nesse ponto, assim como a variável global **h**. Entretanto, a variável global **i** não é acessível a **A** (devido a **A** ter uma variável local de mesmo).

- Quando **B** é chamada por **A**, o parâmetro **w** se torna acessível (com o valor 5), assim como as variáveis locais **j** e **k** e as globais **h** e **i**. Dessa forma, a primeira atribuição dentro de **B** atribui o valor 10 para a variável global **i**.

Parâmetros

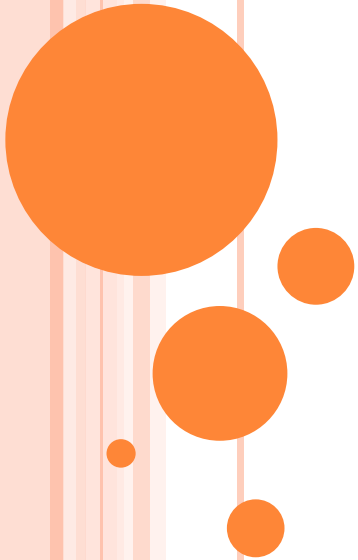
- Funções derivam sua grande utilidade tanto em matemática quanto em linguagens de programação pela sua capacidade de receber parâmetros. Por exemplo, uma função de raiz quadrada não seria muito útil se não pudesse receber um argumento que especificasse o valor cuja raiz quadrada devesse ser calculada.
  - **Definição:** Uma expressão que aparece em uma chamada de função ou em um procedimento é denominada ***argumento***.
  - **Definição:** Um identificador que aparece um uma declaração de função ou em um procedimento é denominado ***parâmetro***.

EXEMPLO EM C/C++

```
int h,i;

void B (int w){
        int j,k;
        i = 2*w;
        w = w+1;
}

void A (int x, int y) {
        bool i,j;
        B(h);
}

int main (){
        int a, b;
        h = 5; a = 3; b = 2;
        A(a,b);
}
```

Na declaração da função **A** temos os *parâmetros* **x** e **y**

Na chamada da função **A** temos os *argumentos* **a** e **b**



EXERCÍCIOS

EXERCÍCIOS

- Pesquise, explique e exemplifique os 5 tipos de variáveis primitivas e 3 tipos de variáveis derivadas (compostas).
- Pesquise sobre a importância de criar códigos modularizados e explique com suas palavras.
- Quais as vantagens da utilização de funções? Exemplifique.
- Quais a diferença entre parâmetro e argumento? Exemplifique.