UFABC

Programação Estruturada

Prof. Fabrício Olivetti de França

Conceitos de Programação

A **computação** está relacionada com a execução de um procedimento bem definido para a solução de um problema.

Não necessariamente está ligado a fazer isso utilizando um computador.

UFABC

Conceitos de Programação

Algoritmo é a descrição da solução do problema computável.

O termo vem de **al-Khwarizmi**, um dos percursores da algebra.

UFABC

Conceitos de Programação

O primeiro algoritmo que se tem conhecimento é o **Algoritmo de Euclides**, utilizado para calcular o **Máximo Divisor Comum**.

UFABC

Conceitos de Programação

Dados dois números inteiros *m*, *n*.

Enquanto o resto *r* = *m* % *n* for diferente de zero:

    Faça *m* = *n*

    Faça *n* = *r*

A solução está em *n*

UFABC

Propriedades de um Algoritmo

1. **Finitude**: um algoritmo deve SEMPRE terminar em um período finito de tempo.

O algoritmo de Euclides é finito, pois *n* é um inteiro que sempre decresce, necessariamente atingindo zero.

UFABC

Propriedades de um Algoritmo

2. **Desambiguidade**: não pode haver ambiguidade em qualquer instrução do algoritmo.

UFABC

Propriedades de um Algoritmo

3. **Entrada**: o algoritmo pode requisitar 0 (zero) ou mais entradas como condições iniciais.

UFABC

Propriedades de um Algoritmo

4. **Saída**: o algoritmo deve ter uma ou mais saídas com a resposta do problema.

UFABC

Linguagens de Programação

Conceitos de Programação

Com o advento dos computadores, passou a ser possível a computação dos algoritmos sem a necessidade de realizar os cálculos manualmente.

Mas para tornar possível a comunicação humano-máquina, foram criadas as linguagens de programação.

UFABC

Cérebro do Computador

A **Unidade de Controle Central** ou **Central Control Unit (CPU)** contém um conjunto de instruções lógicas e artiméticas e é responsável por processar uma sequência de instruções.

UFABC

Cérebro do Computador

Composto de:

- **Unidade de Controle:** decodifica as instruções em comandos de máquina
- **Unidade Lógica e Aritmética:** executa os comandos

UFABC

Conjunto de Instruções

Cada CPU contém um conjunto finito de instruções que permite a definição de algoritmos para serem processados.

UFABC

Conjunto de Instruções

Esse conjunto compreende:

- Instruções
- Registradores
- Endereçamento
- Arquitetura da Memória
- Interrupções
- Entrada e Saída

UFABC

Linguagem de Programação

Linguagem de Programação é uma linguagem bem definida e sem ambiguidades utilizada para se comunicar com o computador.

UFABC

Linguagem de Programação

Inicialmente consistia em um conjunto limitados de **instruções**, pré-definidos pelo processador do computador e um conjunto fixo de espaços de memória para serem utilizadas como **variáveis** do algoritmo.

UFABC

Exemplo Linguagem de Máquina

001000	00001	0000000101011110
Código OP	Endereço	Valor
add	eax	360

UFABC

Algoritmo de Euclides em Ling. Maq.

```
mov esi, 68      # m = 68
mov ebx, 119     # n = 119
jmp .L2          # vai para o passo 2
.L3:
    mov ebx, edx     # n = r
.L2:
    mov eax, ebx
    idiv esi         # EAX = m / n (EAX), EDX = r
    mov esi, ebx     # m = n
    test  edx, edx   # verifica se o resto é zero
    jne .L3          # se teste anterior não zero, vai para L3
```

UFABC

Linguagem de Máquina

Além da dificuldade em entender o que cada instrução faz e a limitação de trabalhar com registradores, o entendimento do fluxo do algoritmo é prejudicado.

Para executar desvios (if-else) e repetições (while/for), usa-se a instrução **jump**, **testes de nulidade**, e marcadores dos blocos de instruções.

UFABC

Linguagem de Máquina

Além disso, tarefas simples como exibir o resultado na tela necessitavam de uma sequência de várias instruções, tornando um processo cansativo e repetitivo.

UFABC

Linguagem de Alto Nível

Para resolver esses problemas, foram criadas linguagens de programação que serviriam como intermediários entre a linguagem de máquina e o programador.

UFABC

Linguagem de Alto Nível

Os objetivos dessas linguagens eram:

- Ter um conjunto de instruções próximas da linguagem natural
- Minimizar o número de instruções para tarefas frequentes
- Ter o máximo de controle sobre o computador, mas sem a necessidade de entendê-lo completamente

UFABC

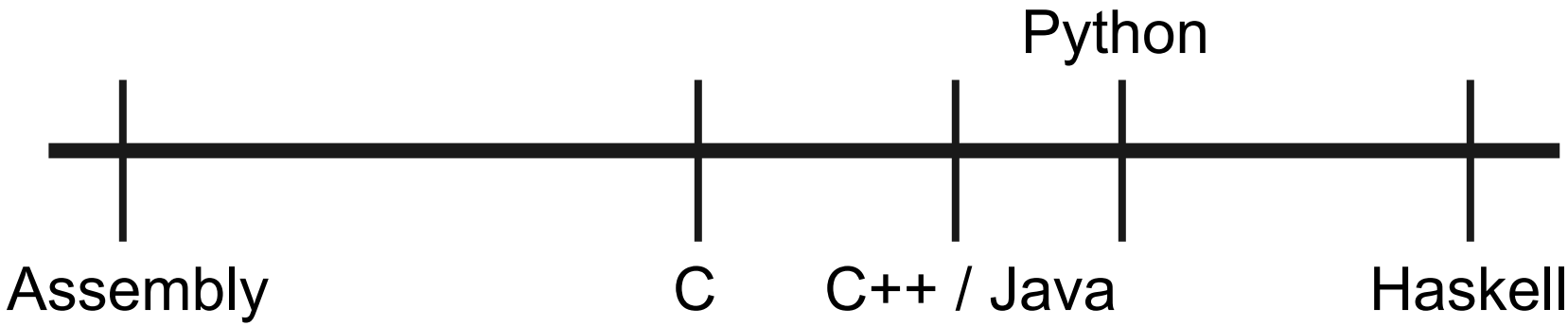
Linguagem de Alto Nível

Não existe uma definição formal de “o quanto” uma linguagem é de alto nível.

Podemos dizer que quanto mais próximo de uma linguagem natural ou quanto menos instruções é necessário para escrever um algoritmo, mais alto nível ela é.

UFABC

Linguagem de Alto Nível



Baixo Nível

Alto Nível

UFABC

Compilador x Interpretador

Compiladas: o código-fonte é traduzido para código de máquina e escrito em um arquivo executável.

Interpretadas: o código-fonte é traduzido para instruções de máquina em tempo de execução.

UFABC

Compilador x Interpretador

Compiladas: permite otimização do código pelo compilador.

Interpretadas: possíveis otimizações do código são mais restritivas.

UFABC

Linguagem de Alto Nível

Nessa disciplina utilizaremos a linguagem **C**:

- Imperativo e estruturado
- Pequeno conjunto de palavras-chaves, operadores, etc.
- Tipagem estática, porém fraca
- Permite criação de tipos de dados abstratos
- Acesso à memória do sistema
- Biblioteca padrão

UFABC

Programação

Imperativa: instruções sequenciais que alteram o estado do sistema.

Funcional: transformação de dados sem alteração de estado.

Orientada a Objeto: define os dados junto com os métodos que os alteram.

UFABC

Programação

Imperativa → estruturada → procedural

Estruturada: divide o algoritmo em blocos organizados

Procedural: modulariza o algoritmo através de procedimentos e funções

UFABC

Tipagem

A declaração de uma variável pode ser:

Estática: o programador deve explicitamente declarar seu tipo e esse tipo não se altera.

Dinâmica: a variável assume um tipo em tempo de execução.

UFABC

Tipagem

Type inference: o programador nem sempre precisa declarar explicitamente o tipo e o compilador define sozinho. No entanto, o tipo ainda é estático!

UFABC

Tipagem

As variáveis podem ser:

Forte: uma variável não pode ser convertida em outro tipo e apenas operações entre variáveis de mesmo tipo.

Fraca: o tipo da variável é flexibilizado nas operações e é possível fazer a conversão de tipos (**casting**).

UFABC

Estruturando o Programa

Algoritmo de Euclides em C

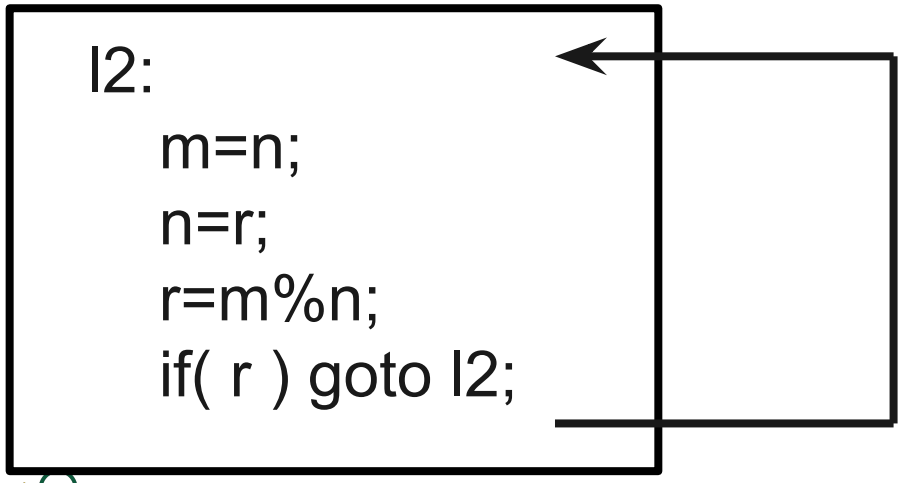
```
int m, n, r;
m = 68;
n = 119;
r = m%n;

l2:
    m=n;
    n=r;
    r=m%n;
    if( r ) goto l2;
```

UFABC

Algoritmo de Euclides em C

```
int m, n, r;
m = 68;
n = 119;
r = m%n;
```



Enquanto r ≠ 0, repete a partir de **l2**

UFABC

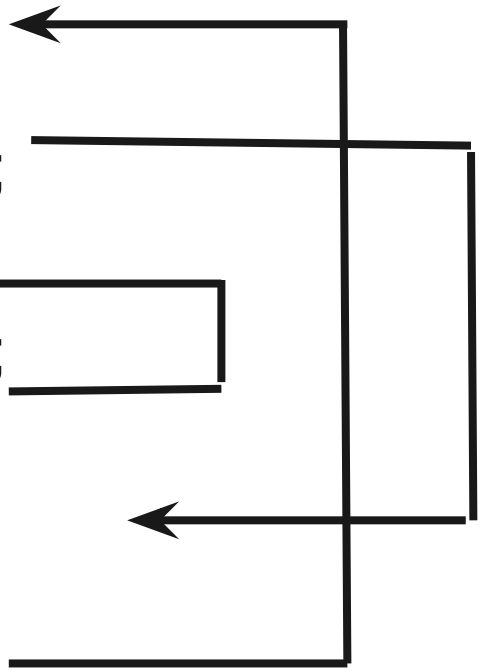
O problema do goto

```
c1:
    if( condicao1 ) goto c3;
c2:
    if( condicao2 ) goto c2;
c3:
    if( condicao3 ) goto c1
```

UFABC

O problema do goto

```
c1:
    if( condicao1 ) goto c3;
c2:
    if( condicao2 ) goto c2;
c3:
    if( condicao3 ) goto c1
```



UFABC

Programação Estruturada

- Divide o algoritmo em blocos de instrução

- Cada bloco descreve um passo do algoritmo:
  - O que deve ser feito x Como deve ser feito

UFABC

Algoritmo de Euclides em C

```
/* valores iniciais */
m = 68;
n = 119;
r = m%n;

/* reduz o problema para um problema mais simples */
while( r!=0 ){
    m = n;
    n = r;
    r = m%n;
}
```

UFABC

Solução de Problemas

Perceberemos ao longo do curso que a solução de muitos problemas consiste em reduzí-los para um problema mais simples em que a solução é trivial.

UFABC

Conteúdo Programático

Conteúdo Programático

http://folivetti.github.io/courses/ProgramacaoEstruturada/

UFABC

Avaliação

2 provas práticas e listas de exercício.

Peso de 40% para cada prova e 20% para as listas.

UFABC

Nota Final	Conceito
>= 9	A
>= 7.5	B
>= 6.5	C
>= 5	D
< 5	F

UFABC

Recuperação

Prova envolvendo todo o conteúdo:

Nota Recuperada = 0.6*REC + 0.4*Nota Final

UFABC

Exemplos

Multiplicação Etíope

Dados *m*, *n* inteiros faça:

Se m for ímpar, some n ao resultado

Divida m por 2, multiplique n por 2

Se m >= 1, vá para o passo (1)

UFABC

Multiplicação Etíope

```
int m, n;
int resultado;
resultado = 0;
repita:
    if( (m%2) != 0 ) resultado += n;
    m /= 2;
    n *= 2;
    if( m > 0 ) goto repita;
```

UFABC

Multiplicação Etíope

```
int m, n;
int resultado;
resultado = 0;
do{
    if( (m%2) != 0 ) resultado += n;
    m /= 2;
    n *= 2;
}while( m > 0 );
```

UFABC

Soma de Dígitos

Para somar os dígitos do número *n* precisamos:

1) Faça total = 0
2) Divida n por 10
3) Some o resto ao total e guarde o quociente em n
4) Se n==0, o resultado está em total, senão retorne ao passo 2

UFABC

Palíndromo

Verificar se um número é palíndromo!

1) Inverta o número *n*
2) Compare o número *n* com ele invertido

UFABC

Inverter o número

n%10 - extrai o dígito mais a direita do número

n/10 - retorna o número sem o dígito mais a direita

n*10 + d - acrescenta o dígito d na posição mais a direita

UFABC

Palíndromo

```
int n, original, inverso;
inverso = 0;
original = n;
repita:
    inverso = inverso*10 + n%10;
    n = n/10
    if( n ) goto repita;
if( original==inverso ) printf(“Palíndromo!\n”);
```

UFABC

Palíndromo

```
int n, original, inverso;
inverso = 0;
original = n;
while( n ){
    inverso = inverso*10 + n%10;
    n = n/10
}
if( original==inverso ) printf(“Palíndromo!\n”);
```

UFABC

Estilos de Programação

Convenções e Estilos

```
If ( … ) {
}
```

Ou

```
if( … )
{
}
```

UFABC

Convenções e Estilos

```
public class Permuter                                          {
    private static void permute(int n, char[] a)               {
        if (n == 0)                                            {
            System.out.println(String.valueOf(a))              ;}
        else                                                   {
            for (int i = 0; i <= n; i++)                       {
                permute(n-1, a)                                ;
                swap(a, n % 2 == 0 ? i : 0, n)                 ;}}}
    private static void swap(char[] a, int i, int j)           {
        char saved = a[i]                                      ;
        a[i] = a[j]                                            ;
        a[j] = saved                                           ;}}
```

UFABC

Convenções

Para facilitar a leitura dos códigos adota-se convenções de escrita de tal forma a forçar o programador a escrever um algoritmo claro, conciso e desambíguo.

Ao longo do curso teremos alguns slides de convenções que serão **OBRIGATORIAMENTE** adotadas nas avaliações.

UFABC

Convenção

Indentação: as instruções dentro dos blocos de instruções devem ser indentadas com 4 ou 8 espaços.

```
{
        Intrução1;
        Instrução2;
        {
                Instrução3;
        }
}
```

UFABC

Convenção

Chaves: as chaves iniciais do bloco de instrução devem ficar na mesma linha do comando de bloco, separados por espaço.

```
if ( condição ) {
        Instruções;
}
```

UFABC

Convenção

Espaço: coloque espaço entre o comando de bloco e a condição, mas não entre o nome de uma função e os parâmetros.

```
if( condição )
potencia( x, y );
```

UFABC

Convenção

Nomes: as variáveis devem ter nomes curtos e descritivos, usar letras minúsculas e substituir espaço por “_”.

quociente; 

nome_aluno; 

nome_do_aluno_matriculado; 

UFABC

Convenção

Nomes: se for uma variável local de uso comum, pode usar nomes bem pequenos.

```
i;    /* inteiro usado como contador */
tmp; /* variável temporária */
mdc; /* máximo divisor comum */
```

UFABC

Convenção

Nomes: constantes são nomeadas com letras maiúsculas, seguindo as outras regras dos nomes de variáveis.

PI

TAMANHO

VALOR_MAX

FORCA_GRAVITACIONAL

UFABC

Convenção

Comentários: preferencialmente apenas no início de cada função ou de um bloco principal. Ela deve descrever o que o bloco faz e não como faz.

Se sentir necessidade de comentar o que uma parte do código faz, reescreva o código.

UFABC

Convenção

Comentários: pode conter múltiplas linhas.

```
/*
 * Esse é um comentário com
 * múltiplas linhas.
 */
```

UFABC

Exercícios para Casa

Exercício 1

Crie um programa que calcule n! (n fatorial). Lembrando que:

n! = n . (n-1) . (n-2) … 1

Faça duas versões:

Uma utilizando apenas **if** e **goto**

Outra utilizando blocos de repetição

UFABC

Exercício 2

Implemente o algoritmo de ordenação Bubble Sort:

1. Dado um vetor **v** de tamanho **N**
2. Para i = 0 até N-1:
    a. Para j = N-1 até i+1:
        i. Se v[j] < v[j-1]:
            1. Troque os valores de v[j] com v[j-1]

Use o código BubbleSort.c como base

UFABC

Exercício 3

Implemente o seguinte algoritmo misterioso:

1. Dados três vetores **v1**, **v2** e **v3** de tamanho **N**
2. Faça **trocou=1**
3. Enquanto **trocou** for igual a 1:
4. Faça **trocou=0**
5. Para i de 0 a N, j de 0 a N e k de 0 a N,
6. Verifique se existe um p <= v2[j] tal que $v1[i]^p = v3[k]$
7. Em caso afirmativo, faça v2[j] = p e faça **trocou=1**

UFABC

Exercício 3

Observação: esse algoritmo não faz nada de útil, mas sua solução será utilizada na próxima aula como motivação.

UFABC