Lógica de Programação

Ricardo Terra

rterrabh (at) gmail.com

CV

- **Nome**: Ricardo Terra
- **Email**: rterrabh (at) gmail.com
- **www**: ricardoterra.com.br
- **Twitter**: rterrabh
- **Lattes**: lattes.cnpq.br/ 0162081093970868

Ph.D. (UFMG/UWaterloo),

Post-Ph.D. (INRIA/Université Lille 1)

Background

Acadêmico : UFLA (desde 2014), UFSJ (1 ano), FUMEC (3 anos), UNIPAC (1 ano), FAMINAS (3 anos)

Profissional : DBA Eng. (1 ano), Synos (2 anos), Stefanini (1 ano)

Conteúdo

Introdução

Arquitetura de um Sistema de Computador

Nessa seção, veremos:

- Arquitetura de um sistema de computação
  - Memória
  - Unidade Central de Processamento (CPU)
  - Periféricos (dispositivos de entrada e saída)
    - Teclado
    - *Mouse*
    - Monitor

Explicando...

Periféricos (dispositivos de entrada e saída)

- **Objetivo**: prover dados ao computador e obter respostas do processamento

Dispositivos

- Teclado e *mouse*
  - Dispositivos de entrada comum. Existe mais algum?
- Monitor
  - Dispositivo de saída comum. Existe mais algum?

Memória

- **Objetivo**: armazenar dados ou programas (sequências de instruções) em um base temporária ou permanente

Dispositivos

- ROM, RAM, HD... Existe mais algum?

Unidade Central de Processamento (CPU)

- Também conhecido como processador
- É a parte de um computador que realiza a maior parte do processamento
- CPU e memória constituem a parte central de um computador, no qual os periféricos serão anexados
  - Veremos a arquitetura Von Neumann

Resumo do que realmente importa

- **Programas** residem na **memória**
- As **instruções** de um programa – soma, divisão, atribuição, laços de repetição, instruções condicionais etc – são executadas pela **CPU**
- Os **dados** necessários para execução do programa são obtidos pelos **dispositivos de entrada** (mouse, teclado, por exemplo) e os **resultados** são apresentados através dos **dispositivos de saída** (monitor, por exemplo)

Introdução

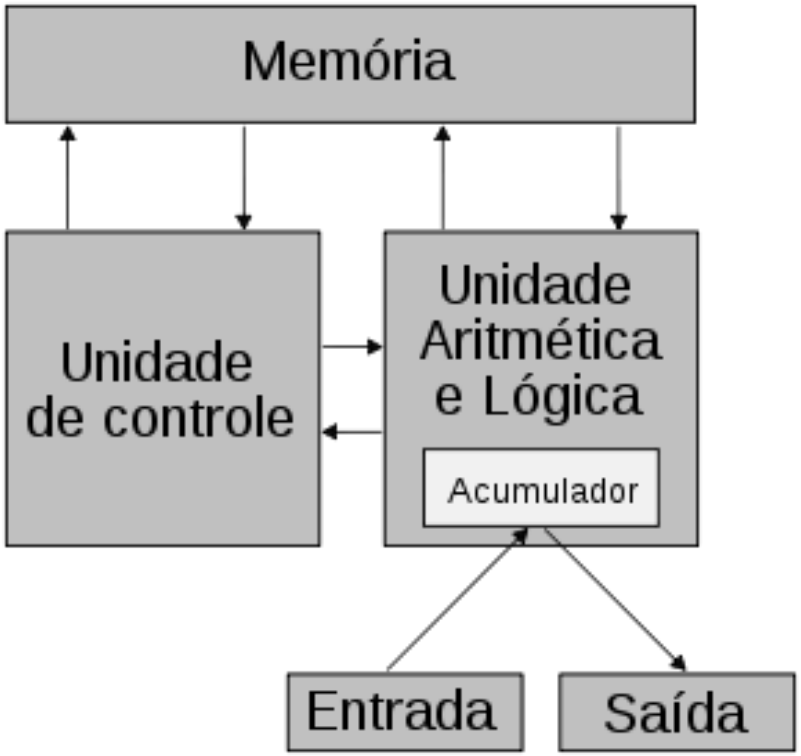
Histórico da Computação

Nessa seção, veremos:

- Histórico da Computação
  - Existem mil modos de abordar esse assunto, mas aqui vamos falar o mais relevante ao curso

Arquitetura Von Neumann

- Von Neumann, temático húngaro, naturalizado norte-americano, propôs nos anos 40 do século XX, um padrão de arquitetura de computadores utilizado até os dias de hoje



Necessidade

- Antigamente, processamentos em geral, como cálculos matemáticos complexos, folhas de pagamento etc
- Atualmente, sistemas bancários, ERP, sistemas de gestão, sistemas *web* etc

Necessidade

- Enfim, o que um computador faz bem?

Necessidade

- Antigamente, processamentos em geral, como cálculos matemáticos complexos, folhas de pagamento etc
- Atualmente, sistemas bancários, ERP, sistemas de gestão, sistemas *web* etc

Necessidade

- Enfim, o que um computador faz bem? **REPETIÇÃO!**

Paradigmas de Programação

- Procedural (imperativo) – Antigamente e atualmente
  - É um paradigma de programação que descreve a computação como ações (instruções) que mudam o estado (variáveis) de um programa. Muito parecido com o comportamento imperativo das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar
- Orientado a objetos – Atualmente
  - É um paradigma de programação que utiliza "objetos" e a interação entre eles para projetar aplicações e programas de computador. Suas técnicas de programação podem incluir características como encapsulamento, polimorfismo e herança. Esse paradigma não era comumente usado no desenvolvimento de aplicações de grande porte até início de 1990. Atualmente, várias linguagens modernas suportam POO

Linguagens – Evolução

- Assembly
- Pascal
- C
- C++
- PHP
- Java
- .NET (C#, J#...)

Introdução

Utilização dos Computadores

Nessa seção, veremos:

- Conceito de programa e sistema
- Linguagem de alto nível e baixo nível
- Compiladores / Interpretadores
- Codificação ASCII
- Áreas de aplicação dos computadores

Programa x Sistema

- Programa
  - Concretização de um algoritmo em alguma linguagem de programação

    ou

    Uma sequência de instruções que descrevem uma tarefa a ser realizada por um computador

- Sistema
  - Grupo de componentes de hardware e sistemas de software, projetados e montados para executar uma função específica ou grupo de funções

Níveis de linguagem

- Representa o nível de abstração da linguagem
- Quanto mais longe do código de máquina (ou, em outras palavras, quanto mais próximo à linguagem humana) mais alto é o seu nível
- Níveis:
  - Alto: Pascal, COBOL, Java...
  - Médio: C, C++...
  - Baixo: Assembly...

Compiladores X Interpretadores

- Interpretador lê o código-fonte linha a linha, executando a instrução específica daquela linha
- Compilador lê o programa inteiro, converte-o em um código-objeto (ou código de máquina) de modo que o computador consiga executá-lo diretamente

- *Qual é mais rápido?*

Codificação ASCII

- Códigos ASCII representam texto em computadores, equipamentos de comunicação e outros dispositivos que utilizam texto
- Por exemplo, o computador não sabe o que é **`A`**, mas se você informar **`65`** e dizer que é um caractere, ele saberá que refere-se ao caractere **`A`**
  - Na verdade, na verdade mesmo, ele nem sabe o que é **`65`**, sabe o que é **`01000001`**
- Veja a tabela ASCII em: http://www.asciitable.com

Áreas de aplicação de computadores

- Praticamente TODAS, alguns exemplos:
  - Sistemas de grande porte
  - Sistemas *web*
  - Banco de dados
  - Bolsa de valores
  - Processamentos pesados
  - Estatística
  - ...

Conteúdo

Lógica Proposicional

Introdução

Introdução

- Antes de entrarmos com desenvolvimento de algoritmos, é interessante aprendermos a organizar nossas ideias e saber como organizá-las
- Nesse intuito, é apresentada um pouco da lógica proposicional para que vocês treinem formalização em linguagens não naturais
  - No caso, uma linguagem matemática
  - No futuro, Portugol e C
- Este capítulo servirá como uma base forte para a nossa disciplina, cujos resultados começarão a logo perceber

Problema do Bezerro

Em uma fazenda existem 20 vacas e seus 21 bezerros. Nenhuma possui dois bezerros.

É possível existir tal fazenda?

Paradoxo do Barbeiro

Em uma determinada cidade:

- Todo homem que é capaz de se barbear, o faz
- O barbeiro da cidade faz a barba daqueles – e apenas daqueles – que não são capazes de se barbear

É possível existir tal cidade?

Problema do Peão e do Cavaleiro

Peões mentem, cavaleiros dizem a verdade.

Em um grupo, você pergunta para a pessoa **A**: "Quantos de vocês são cavaleiros?". **A** responde, mas você não entende a resposta.

Você então pergunta para a pessoa **B**: "O que ele disse?" e a resposta é: "Ele disse que exatamente dois de nós somos peões". Então, **C** comenta: "Ele (**B**) está mentindo".

C é um cavaleiro ou um peão?

Lógica Proposicional

Material

Material

- Iremos utilizar o material do Prof. Dr. Newton José Vieira[a], um dos melhores e mais renomados professores da Universidade Federal de Minas Gerais (UFMG)
- Vejam materiais:
  - **`logicaproposicional_apostila.pdf`**
  - **`logicaproposicional_slides.pdf`**

[a] www.dcc.ufmg.br/~nvieira

Conteúdo

Algoritmos e Programação Estruturada

Conceitos Importantes

Nessa seção, veremos:

- Algoritmos e Programação Estruturada
  - Algoritmo
  - Estrutura de dados
  - Programação estruturada

Algoritmo

- "Um **algoritmo** é a descrição de um padrão de comportamento, expressado em termos de um repertório *bem definido* e *finito* de ações "primitivas", das quais damos por certo que elas podem ser *executadas*."

  ou

  "Um **algoritmo** é, em outras palavras, uma *norma executável* para estabelecer um certo efeito desejado, que na prática será geralmente a obtenção de uma solução a um certo tipo de problema."

Exemplo – Algoritmo para descascar as batatas para o jantar

```
"traga a cesta com batatas do porão";
"pegue a panela no armário";
se ("saia é clara") {
    "coloque avental";
}
enquanto ("número de batatas é insuficiente") {
    "descasque uma batata";
}
"devolva a cesta ao porão";
```

Estrutura de dados

- Representam as informações do problema a ser resolvido
- A formulação do algoritmo e a definição das estruturas de dados estão intimamente ligadas

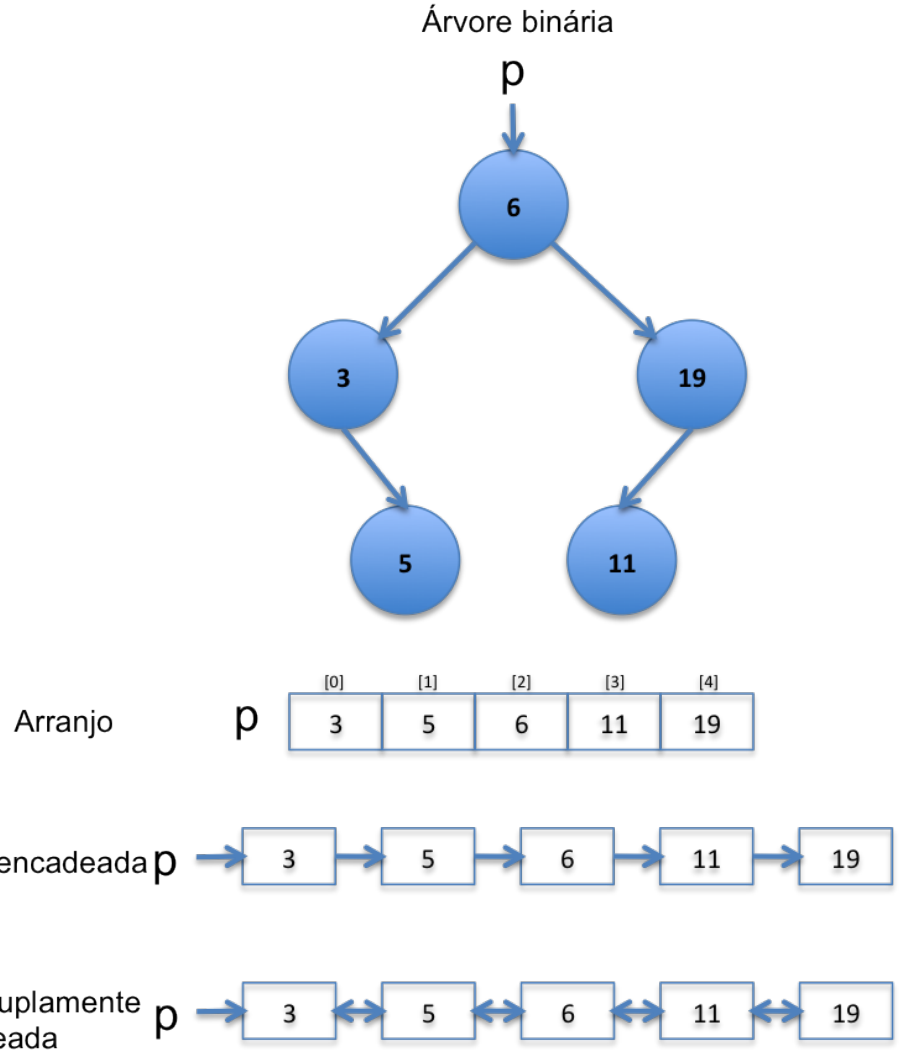
Enfim

- "decisões sobre a estruturação de dados não podem ser feitas sem conhecimento dos algoritmos aplicados a eles"

  e

  "a estrutura e a escolha dos algoritmos depende muitas vezes fortemente da estrutura dos dados"

Estrutura de dados



Qual das estruturas de dados anteriores é melhor para:

- retornar os elementos ordenados?
- pesquisar um elemento?
- acrescentar um elemento?
- inserir um elemento?
- remover um elemento?
- alterar o valor de um elemento?

Programa

- Programas são formulações concretas de algoritmos abstratos, baseados em representações e estruturas específicas de dados
- Programar é basicamente construir algoritmos

Programação Estruturada

- Basicamente, uma metodologia de projeto de programas visando:
  - Facilitar a escrita dos programas
  - Facilitar a leitura (o entendimento) dos programas
  - permitir a verificação *a priori* dos programas
  - facilitar a manutenção e modificação dos programas
- Todos os programas podem ser reduzidos a apenas três estruturas:
  - sequência
  - decisão
  - iteração

Programação Estruturada – Exemplo

```
1  principal() {
       inteiro i = 1; /* Declarando um inteiro */
3      imprima("Números pares até 100:");
       enquanto (i <= 100) {
5          se (i mod 2 == 0) {
               imprima(i);
7          }
           i = i + 1;
9      }
   }
```

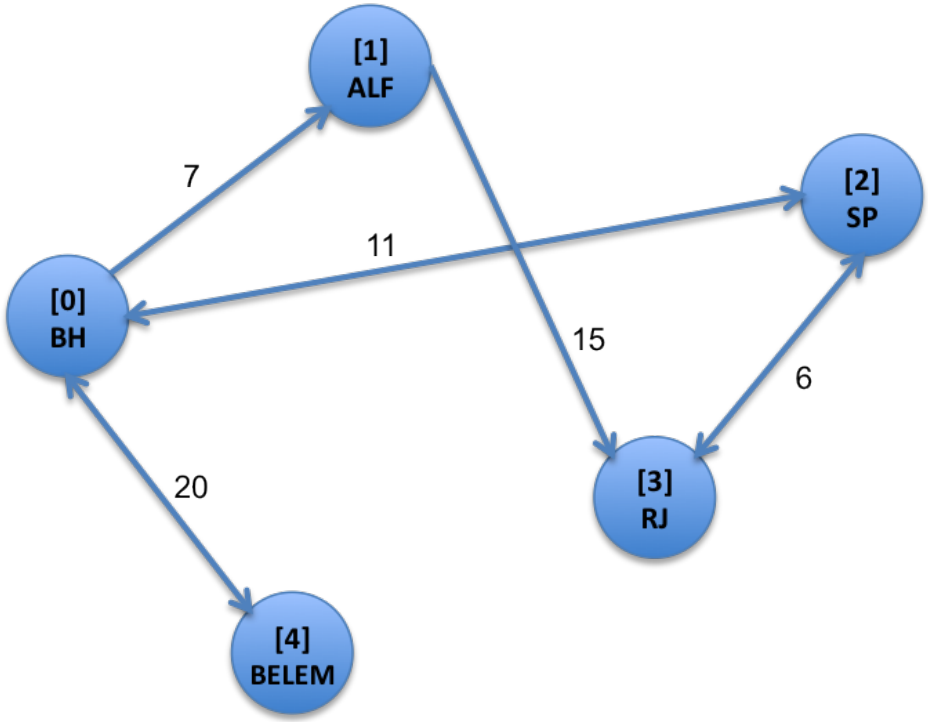
Observações

- Observe a sequência, a decisão e a iteração
- Não se preocupe com sintaxe, veremos o **PortuCol** logo

Exercícios Propostos

- Escreva um algoritmo para:
  - Ler um número e dizer se é par ou ímpar
  - Ler dois números $\mathbf{x}$ e $\mathbf{y}$ e imprimir $\mathbf{x^y}$
  - Ler um caractere e dizer se é vogal ou consoante

Exercício Proposto



Dada a modelagem acima, pense em um algoritmo para:

- descobrir o menor caminho entre duas cidades (*Dijsktra*)
- resolver o problema do caixeiro viajante

Conteúdo

Portucol

Introdução

Nessa seção, veremos:

- **Portucol**
  - Introdução à linguagem (adaptação do **Portugol** para C)
  - Sintaxe e semântica dos comandos básicos
  - Leitura e escrita de algoritmos

Portucol – Introdução

Portucol

- Pseudolinguagem de programação inspirada no conhecido **Portugol**
- Objetivo:
  - Obter uma notação para algoritmos a ser utilizada na *definição*, na *criação*, no *desenvolvimento* e na *documentação* de um programa

Regras Básicas

- Os comandos devem ser escritos sempre em caixa baixa
- Ao final de cada comando deverá ser colocado um ponto e vírgula (`;`)
- Os nomes das variáveis e identificadores devem:
  - ser compostos por letras (caixa baixa) e números, **mas** devem começar com uma letra
  - Ex: **`idade`**, **`nome`**, **`a4`**...
- O algoritmo deve ser escrito de forma estruturada e indentada

Tipos de Dados

- **Inteiro**
  - qualquer número inteiro, negativo, nulo ou positivo
  - Ex: -5, 0, 12 ...
- **Real**
  - qualquer número real, negativo, nulo ou positivo
  - Ex: -5, 30.5, 0, 40.1, -5.2 ...
- **Caractere**
  - um caractere alfanumérico, exemplo: `A', `a', `0' ...
- **String**
  - conjunto de caracteres alfanuméricos
  - Ex: "Lógica", "LP", "Carnaval" ...
- **Lógico**
  - verdadeiro ou falso

Declaração de variáveis

- `inteiro idade;`
- `real preco;`
- `caractere inicial;`
- `String nome;`
- `lógico flag;`

Comentários

- É recomendado e muito importante inserir comentários em seus códigos
- Os comentários devem estar entre **/*** e ***/**

Exemplo

```
principal() {
    inteiro i = 1; /* Declarando um inteiro */
    imprima("Números pares até 100:");
    enquanto (i <= 100) {
        se (i mod 2 == 0) { /* se o resto for 0 é par */
            imprima(i);
        }
        i = i + 1; /* incrementa contador */
    }
} /* fim da funćão principal*/
```

Operadores

- `=` atribuição
- `+` soma
- `-` subtração
- `*` multiplicação
- `/` divisão
- `%` resto (pode ser **`mod`** também)

Funções mais comuns

- **`sqrt`** raiz quadrada ex: **`sqrt(4)`**
- **`pow`** exponenciação ex: **`pow(3,2)`** (equivale a 3^2)

Operadores Relacionais

- `==` comparação
- `!=` diferente
- `<=` menor ou igual
- `>=` maior ou igual
- `>` maior
- `<` menor

Operadores Lógicos

- `e` conjunção
- `ou` disjunção
- `não` negação

Precedência

1. `( )`
2. funções
3. `* /`
4. `+ -`
5. `==, !=, <, <=, >, >=`
6. `não`
7. `e`
8. `ou`

Comando de entrada

- **leia**
  - Faz a leitura de qualquer variável
  - Ex: **leia(x)**
- **imprima**
  - Faz a impressão de qualquer variável e texto
  - Exemplos:
    - **imprima(x)**;
    - **imprima("O valor digitado foi $x")**

Portucol

Comandos Básicos

Comandos condicionais - `se`

- Sintaxe do comando **`se`**

```
se ( condicao ) {
    comando1;
    comando2;
    comando3;
}
```

Observações:

- Os parênteses que envolvem a condição são **OBRIGATÓRIOS**
- A condição deverá retornar um tipo lógico
- Os comandos somente serão executados se a condição for verdadeira

Comandos condicionais - `se`

- O uso dos braços **NÃO** é obrigatório caso seja apenas um único comando

- Porém, a boa prática recomenda a utilização de braços independente do número de comandos
  - Melhor indentação do código

```
se ( verdade ) comando;
```

equivale a:

```
se ( verdade ) {
    comando;
}
```

Comandos condicionais - `se`

- Como faço o conhecido:

  se *verdade* então . . . senão . . .

```
se ( condicao ) {
    comando1;
    comando2;
    comando3;
} senão {
    comando4;
    comando5;
    comando6;
}
```

Pergunta-se

Quais comandos serão executados se a condição for verdadeira? E se for falsa?

Comandos condicionais - se

```
1  principal (){
       inteiro a, b;
3      leia(a);
       leia(b);
5      se ( a >= b ) {
           imprima("$a e maior ou igual a $b!");
7      } senão {
           imprima("$a e menor que $b!");
9      }
   }
```

Comandos condicionais - `se`

- Comando **`se`** podem ser aninhados

```
1  principal (){
       inteiro nota;
3      leia(nota);
       se ( nota >= 90 ) {
5          imprima("Nota A");
       } senão se ( nota >= 80 ) {
7          imprima("Nota B");
       } senão se ( nota >= 70 ) {
9          imprima("Nota C");
       } senão {
11         imprima("Reprovado");
       }
13 }
```

Exemplo se

```
principal (){
    caractere op;
    leia(op);
    se (op == '+' ou op == '-') {
        imprima("Operador de Baixa Prioridade");
    } senão se (op == '/' ou op == '*') {
        imprima("Operador de Alta Prioridade");
    } senão {
        imprima("Operador Inválido");
    }
}
```

Exercício de Fixação 01

Crie um algoritmo que leia dois números inteiros e imprima a soma, a subtração, a multiplicação, a divisão e o resto da divisão entre esses dois números.

Observe que o valor do segundo número não deve ser 0, portanto faça um **`se`** para evitar isto.

Exercício de Fixação 02

Crie um algoritmo que leia os dois catetos de um triângulo (números reais **a** e **b**) e calcule a hipotenusa:

$$h^2 = a^2 + b^2$$

Comandos repetição (`enquanto`, `faça...enquanto` e `para`)

- Comandos de repetição são utilizados para repetir um bloco de código
- Veremos três comandos de repetição:
  - **`enquanto`**
  - **`faça...enquanto`**
  - **`para`**

Comandos repetição - `enquanto`

- O comando **`enquanto`** é utilizado para repetir um bloco de acordo com uma condição
- É considerado um *loop* de pré-teste
  - Isto é, testa a condição antes de executar o bloco

Sintaxe:

```
enquanto ( condicao ) {
    comando1;
    comando2;
    comandoN;
}
```

Comandos repetição - `enquanto`

```
1  principal (){
       inteiro i = 0;
3
       enquanto ( i < 10 ) {
5          imprima( i );
           i = i + 1;
7      }
   }
```

Pergunta-se

Qual a saída?

Exercício de Fixação

Faça um algoritmo que liste todos os números de 0 a 100 divisíveis por 3

Comandos repetição - `faça...enquanto`

- O comando **`faça...enquanto`** é semelhante ao **`enquanto`**, contudo é um comando de repetição de pós-teste
  - Isto é, somente ao final da execução do bloco que se verifica a condição

- Geralmente, é utilizado quando se deseja testar a condição somente a partir da segunda iteração
  - Por exemplo, uma leitura da opção de um menu. Pede para digitar uma primeira vez. **Somente** se não digitar uma opção válida que pede para digitar novamente

Sintaxe:

```
faça {
    comando1;
    comando2;
    comandoN;
} enquanto ( condicao );
```

Observe o ponto-e-vírgula após os parênteses da condição. Não o esqueça!

Comandos repetição - `faça...enquanto`

```
principal () {
    inteiro i = 0;

    faça {
        imprima( i );
    } enquanto ( 1 != 1 );
}
```

Pergunta-se

Qual a saída?

Exemplo

```
principal () {
    inteiro i;
    faça {
        imprima("Digite um numero entre 0 e 10: ");
        leia(i);
    } enquanto ( i < 0 ou i > 10 );

    imprima(""Numero digitado: $i");
}
```

Pergunta-se

Qual a saída?

Exercício de Fixação

Faça um algoritmo que estipule um número mágico (entre 0 e 100) e peça para o usuário digitar números até que ele acerte.

Quando não acertar, diga se o número mágico é menor ou maior que o número que ele digitou.

Por exemplo:
Número mágico estipulado: 7
1a tentativa: 90 (número mágico é menor)
2a tentativa: 4 (número mágico é maior)
3a tentativa: 7 (parabéns!)

Comandos de repetição - `para`

- Comando de repetição mais poderoso do Portucol
- É composto por:
  - Inicialização: executado uma única vez no início do *loop*
  - Condição: executado sempre antes de cada iteração. Se verdadeira, o bloco é executado. Se falsa, é finalizado
  - Operação : executado sempre ao término de cada iteração

Sintaxe

```
para ( inicializacao ; condicao ; operacao ) {
    comando1;
    comando2;
    ...
    comandoN;
}
```

Exemplo

```
principal () {
    inteiro i;

    para (i = 0; i < 10; i = i+1) {
        imprima(i);
    }
}
```

Pergunta-se

Qual a saída?

Sintaxe

- No laço **para**, a *inicialização*, *condição* e *operação* são todas opcionais

Exemplo

```
1  principal () {
       inteiro i;
3
       para ( ; ; ) {
5          imprima(i);
       }
7  }
```

Pergunta-se

Qual a saída?

Comandos de repetição - `para`

- Podemos ter um **`para`** dentro de outro, e outro dentro de outro, e outro dentro de outro...

```
1 principal () {
      inteiro i, j;
3
      para (i = 0; i <= 2; i = i+1) {
5         para (j = 0; j < 2; j = j+1) {
              imprima("$i $j")
7         }
      }
9 }
```

Pergunta-se

Qual a saída?

Comandos de repetição - `para`

- Um comando **`para`** pode ter várias inicializações, uma condição complexa e várias operações

Exemplo

```
1  principal () {
       inteiro i, d;
3
       para (i = 1, d = 2 * i; i <= 10 ou d == 22; i = i+1, d = i * 2) {
5          imprima("$i $d");
       }
7  }
```

Comando abandone

- Inserido dentro de um bloco de repetição (pode também ser **enquanto** ou **faça...enquanto**)
- Caso seja executado, o bloco de repetição é finalizado

Exemplo

```
1  principal () {
       inteiro i = 0;
3
       para (; i < 10; i = i + 1) {
5          se (i == 3) {
               abandone;
7          }
           imprima(i);
9      }
   }
```

Comando `continue`

- Inserido dentro de um bloco de repetição
- Caso seja executado, a iteração atual do bloco de repetição é interrompida e parte para a próxima iteração

Exemplo

```
principal () {
    inteiro i;

    para (i = 0; i < 10; i = i + 1) {
        se (i == 3 ou i == 5) {
            continue;
        }
        imprima(i);
    }
}
```

Comando de repetição - `para`

- Logicamente, pode-se utilizar **`abandone`** e **`continue`** conjuntamente

Exemplo

```
principal () {
    inteiro i, j;

    para (i = 0; i < 3; i = i + 1) {
        se (i == 1) {
            continue;
        }
        para (j = 0; j < 2; j++) {
            imprima("$i $j");
            abandone; /* abandona sempre o laço mais interno */
        }
    }
}
```

Exercício de Fixação

Faça um algoritmo que leia diversas idades e depois exiba a média.

Duas considerações:
- O algoritmo deverá ignorar idades acima de 120 anos;
- O algoritmo só para de ler idades quando receber o valor **−1**.

Funções importantes

- **`tamanho`** retorna o tamanho de um *string* ou arranjo. Por exemplo, suponha que **`str`** contenha "ANA", assim **`tamanho(str)`** retornará 3
- **`encerra`** encerra o programa abruptamente

Importante

- Ao ler um *string*, você pode acessar cada posição utilizando colchetes **[ ]**
  - Índices vão de 0 a tam-1

Exemplo

```
principal() {
    string str = "ANA";
    inteiro i, contador = 0;
    para ( i=0; i < tamanho(str); i = i + 1 ) {
        se ( str(i) == 'A' ) {
            contador = contador + 1;
        }
    }
    imprime("Numero de A's: $contador");
}
```

Portucol

Arranjos

Arranjos - Conceito

- Arranjos – também conhecidos como vetor, *array* etc – são coleções de um mesmo tipo em sequência
- Arranjos podem ser de qualquer tipo visto (inteiro, real, caractere, *string* ou lógico)
- Pode se ter um arranjo de inteiros ou um arranjo de caracteres ou um arranjo de arranjo de reais
  - Contudo, não se pode ter um arranjo que contenha inteiros e *strings*

Arranjos - Declaração

- Declaração:

```
inteiro notas(5); /* Arranjo de 5 inteiros */
caractere letras(5); /* Arranjo de 5 caracteres */
```

- Assim como variáveis comuns, os elementos do arranjo **não** são inicializados automaticamente. Contudo, você pode declarar já inicializando:

```
inteiro notas(5) = {4,6,6,9,8};
caractere letras(5) = {'A', 'E', 'I', 'O', 'U'};
```

Arranjos - Declaração

- Um arranjo de tamanho `n`, tem suas posições indexadas de `0` a `n-1`
- Para obter o tamanho de um arranjo, basta chamar a função **`tamanho`**

Exemplo

```
principal() {
    inteiro v(5) = {1, 2, 3, 4, 5};
    inteiro i;
    para ( i=0; i < tamanho(v); i = i + 1 ) {
        imprima( v(i) );
    }
}
```

Arranjos - Acesso aos elementos

- Arranjos permite recuperar ou alterar qualquer um de seus elementos
- Os arranjos sempre iniciam-se na posição 0
  - Isto indica que ele termina em uma posição inferior ao tamanho (`n-1`)

Exemplo

```
1  principal() {
       caractere v(5) = {'A', 'E', 'Y', 'O', 'U'};
3
       imprima( "A primeira vogal é $v(0)" );
5
       v(2) = 'I';
7
       imprima( "A última vogal é $v(4)" );
9  }
```

Arranjos Multidimensionais

- Pode-se criar um arranjo de arranjos
- O mais comum é o bidimensional que vemos como uma matriz
- A declaração é somente acrescentar o número de colunas
- Por exemplo: **inteiro** `matriz[4][3]` declara-se uma matriz de 4 linhas e 3 colunas

```
matriz[4][3] = {{1,0,0},{0,1,2},{2,3,4},{0,6,7}};
```

Representação:

1	0	0
0	1	2
2	3	4
0	6	7

Arranjo Bidimensional – Exemplo

- Declarando e inicializando uma matriz 3x2 e imprimindo a soma dos valores

```
principal() {
    inteiro v(3)(2) = { {1, 4}, {5, 6}, {9, -11} };
    inteiro i, j, soma = 0;
    para ( i=0; i < tamanho(v); i = i + 1 ) {
        para ( j=0; j < tamanho(v(i)); j = j + 1 ) {
            soma = soma + v(i)(j);
        }
    }
    imprima( "Soma da matriz: $soma" );
}
```

matriz

1	4
5	6
9	-11

Exercício de Fixação 01

Crie um algoritmo que declare e leia cada posição de um arranjo de inteiros de tamanho 5 e imprima a soma e a média aritmética

Exercício de Fixação 02

Crie um algoritmo que declare e leia cada posição de um arranjo de reais de tamanho 10 e imprima a soma total deste arranjo considerando a seguinte fórmula:

```
soma = 1 * array[0] +  2 * array[1] + ... +
       9 * array[8] + 10 * array[9]
```

Portucol

Funções

Função

- Um algoritmo é uma coleção de funções
- Uma das funções deve se chamar **`principal`**
  - por onde começa a execução do algoritmo
- Uma função pode:
  - receber parâmetros
  - declarar variáveis locais
  - conter instruções executáveis
  - retornar um valor

Função

- O comando **retorna** efetua o retorno (término) da função
- Uma função pode ou não ter um retorno
- Uma função pode ou não ter parâmetros formais

Exemplos

```
m1() { ... } /* sem retorno e sem parâmetros formais */
m2(real x) { ... } /* sem retorno e com um par. formal */
inteiro m3() { ... } /* com retorno e sem parâmetros formais */
inteiro m4(caractere c, inteiro i) { ... } /* com retorno e 2 par. */
```

Função

- **Sintaxe:**

  retorno nome (< param { , param} >) { *corpo* }

Exemplos

```
1  imprimir() { ... }

3  inteiro dobro(inteiro x) { ... }

5  real somar(real a, real b) { ... }

7  listar(inteiro notas()) { ... }
```

Função

- Retorno de funções
  - Uma função pode retornar valores de qualquer tipo
  - Uma função que retorna nada, não declara retorno
  - A expressão que segue o **`retorna`**, é o valor retornado pela função
    - não é necessário parênteses

Função

- Término de uma função
  - Ao encontrar a chave de fechamento
  - Ao ser retornada (**`retorna`**)

Exercícios de Fixação

- Implementar as funções:
  - `soma`
  - `fatorial`
  - `escreveMaior`
  - `retornaMenorElemento`
  - `retornaMaiorElemento`
  - `retornaMedia`

Exercício - Soma

```
1  principal() {
       real x, y, total;
3      leia(x);
       leia(y);
5      total = soma(x,y);
       imprima("$x + $y = $total");
7  }

9  real soma (real x, real y) {
       retorna x+y;
11 }
```

Exercício - Fatorial

```
1  principal() {
       inteiro n;
3      leia(n);
       imprima( fat(n) );
5  }

7  inteiro fat (inteiro n) {
       inteiro i, resultado = 1;
9      se (n == 0 ou n == 1) {
           retorna 1;
11     }
       para ( i=2; i <= n; i = i+1 ) {
13         resultado = resultado * i;
       }
15     retorna resultado;
   }
```

Exercício - Escreve Maior

```
principal() {
    inteiro n;
    leia(n);
    retornaMaior( n );
}

retornaMaior (inteiro n) {
    inteiro i;
    para (i = n; i >= 1; i = i - 1) {
        imprime( i );
        se ( i != 1 ) {
            imprime(" > ");
        }
    }
}
```

Exercício - Retorna Menor Elemento

```
1  principal() {
       inteiro v(5) = { 2, 4, 5, 1, 3 }, min;
3      min = menorElemento(v);
       imprima( "Menor elemento: $min" );
5  }

7  inteiro retornaMenorElemento(inteiro v()) {
       inteiro i;
9      inteiro menor = v(0);
       para (i = 1; i < tamanho(v); i = i+1) {
11         se (v(i) < menor) {
               menor = v(i);
13         }
       }
15     retorna menor;
   }
```

Conteúdo

Algoritmos com Qualidade

Introdução

Nessa seção, veremos:

- Boas práticas de programação
- Metodologia para o desenvolvimento de algoritmos

Algoritmos com Qualidade

Boas práticas de programação

1. Algoritmos devem ser feitos para serem lidos por seres humanos

- Outras pessoas poderão ter que corrigí-lo, adaptá-lo, modificá-lo...
- Ainda, talvez você mesmo tenha que entendê-lo depois de um ano que escreveu

2. Comente seu código

- Comentário é sinal de saber desenvolver
- Facilita o entendimento do código
- Existem coisas que você faz que – poucos segundos depois – nem lembra como fez

3. No entanto, comentários devem acrescentar algo não apenas frasear os comandos

- Comentários **não** dizem o que está sendo feito, mas sim, *por quê*
- Exemplo de comentário que não se deve fazer:
  - `imprima( total ); /* imprime o total */`
- Exemplo de comentário que se deve fazer:
  - ```
    /* implementação da RN021 */
    se ( taxaJuros < 3.0 e selic > taxaJuros ) {
        ...
    }
    ```
    ```

4. Use comentários no prólogo de funções

- É interessante que funções tenham comentários contendo:
  - um descrição sucinta do que faz
  - autor
  - data de escrita

5. Utilize espaços em branco para melhorar a legibilidade

- melhoram substancialmente a aparência de algoritmos
- principalmente no início do algoritmo, pois pode ser que haja a necessidade de mais variáveis

6. Variáveis com nomes representativos

- Nomes de variáveis devem identificar o que representam
  - Por exemplo, `x = y + z` é muito menos claro que `preco = custo + lucro`
- Outras dicas:
  - Para variáveis de iteração, use `i`, `j`, `k`...
  - Para variáveis auxiliares, use `aux`

7. Um comando por linha

- Vários comandos por linha, prejudicam a legibilidade

8. Utilize parênteses para aumentar a legibilidade e prevenir erros bobos

- Por exemplo:

```
A * B * C / (D * E * F)
```

seria ainda mais claro se fosse:

```
(A * B * C) / (D * E * F)
```

9. Identação é imprencidível

- Melhora a legibilidade
- Apresenta a estrutura lógica do algortimo
- Deve seguir o padrão (novos blocos são identados)

10. Alteração no algoritmo → Alteração no comentário

- Sempre que um algoritmo for alterado, os comentários também devem ser alterados, não apenas os comandos
- Antes não comentar do que deixar um comentário errado

Algoritmos com Qualidade

Metodologia para o desenvolvimento de algoritmos

Algoritmos com Qualidade – Metodologia para o desenvolvimento de algoritmos

Uma boa sequência a ser seguida:

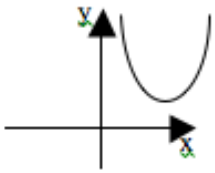
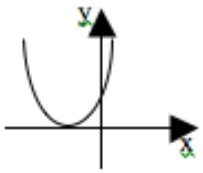
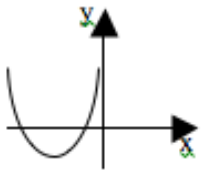
- **Passo 1**: leia o enunciado do problema até o final (sem fazer nenhum tipo de anotação)
- **Passo 2**: repita o "*Passo 1*" até entender
- **Passo 3**: levantar e analisar todas as saídas do problema
- **Passo 4**: levantar e analisar todas as entradas do problema
- **Passo 5**: levantar as variáveis necessárias
- **Passo 6**: pensar como transformar entradas em saídas
- **Passo 7**: testar cada parte do algoritmo (simule entradas e confira saídas)
- **Passo 8**: testar o algoritmo como um todo (simule entradas e confira saídas)

Exercício de Fixação

De acordo com a metodologia, desenvolva um algoritmo que leia as três variáveis da equação de 2^o grau (**a**, **b** e **c**) e imprima se a parábola gerada pela equação toca o eixo das abcissas ou não. Se tocar, imprima também as raízes resultantes

Fórmula de Baskara: $x = \frac{-b \pm \sqrt{\Delta}}{2 \cdot a}$, *sendo a diferente de 0.*

$$\Delta = b^2 - 4 \cdot a \cdot c$$

Se Δ < 0, a parábola não toca o eixo x.	Se Δ = 0, só existe uma raiz.	Se Δ > 0, existem duas raízes.
		

Referencias


Angelo de Moura Guimarães e Newton Albert de Castilho Lages.
Algoritmos e Estruturas de Dados.
Editora LTC, 1994.


Victorine Viviane Mizrahi.
Treinamento em Linguagem C.
Prentice-Hall, 2 edition, 2008.


Newton José Vieira.
Lógica aplicada à computação.
http://homepages.dcc.ufmg.br/ nvieira/cursos/ldt/notas-de-aulas/logica1.pdf, 2007.