

Introdução à programação

Conteúdo

Páginas

Índice	1
Prefácio	2
Programar	2
Como programar	4
Definições sobre Lógica de Programação	7
História da Programação	10
Lógica	11
Aprendizagem	13
Algoritmos	14
Estruturas de manipulação de dados	19
Pseudo-código	25
Expressões em pseudo-código	27
Orientação a objectos	28
Exercícios	30
Soluções dos exercícios	31
Anexo: Linguagens de programação	33
Anexo: Passagem para a linguagem de programação escolhida	36
Bibliografia	36
















Referências

Fontes e Editores da Página	38
Fontes, Licenças e Editores da Imagem	39



Licenças das páginas

Licença	40
---------	----

Índice

- Capa
- Prefácio 
- Programar, o que é a programação? 
- Como programar 
- Estrutura interna de um computador
- Processamento de dados
- Definições sobre Lógica de Programação 
- História da Programação 
- Lógica 
- Aprendizagem: o "padrão eficaz" 
- Como Aprender
- O Padrão Eficaz
- Algoritmos 
- Estruturas de manipulação de dados 
- Estruturas básicas
- Estruturas de Controlo
- Funções
- Pseudo-código 
- Expressões em pseudo-código 
- Orientação a objectos 
- Exercícios 
- Soluções dos exercícios 
- Bibliografia e ligações externas 

Anexo

- Linguagens de programação 
- História e evolução das linguagens
- Igualdades e diferenças nas linguagens
- Passagem para a linguagem de programação escolhida 

Leia também

- Algoritmos e Estruturas de Dados - para explicações mais detalhadas e avançadas
- Programação em GUI
- Programar em C
- Programação orientada a objetos
- Python



Este livro foi eleito pelos colaboradores como um dos melhores do Wikilivros. Para mais informações, consulte a página de votações

Prefácio

Com a evolução da tecnologia, a cada dia mais pessoas têm acesso a um computador, seja em casa, na escola, no trabalho. E para alguns mais curiosos, surgem perguntas como: *"como eles fazem isto?"*, *"será que posso fazer ou aprender?"*, *"como será um computador por dentro?"*.

Muitos já se aventuraram na procura de respostas das suas auto-questões, porém nem sempre é fácil achar o que se pretende. Aqui o leitor tem uma oportunidade de perceber como funciona tudo isto.

O objetivo deste livro é servir como base para qualquer pessoa que queira mergulhar, ou apenas conhecer, o maravilhoso mundo da programação, mesmo que saiba pouco ou nada sobre o assunto.

Este livro também pode servir como uma forma de enriquecimento cultural sobre temas já esquecidos pelo tempo, pois ele aborda assuntos desde a arquitetura de processadores e computadores, cálculos computacionais, lógica e matemática até uma breve história sobre as linguagens de programação e programação básica de algoritmos.



O Morse é uma sequência lógica muito parecida com o código binário

Este guia também se destina aos que querem participar na atividade comunitária de produção de software livre, mas não receberam formação técnica do gênero, motivos para o fazer é que não faltam.

Programar

Motivação

Nos dias que correm, não saber trabalhar com computadores é considerada iliteracia (analfabetismo) e o custo por não saber utilizar um computador pode ser caro.

Quando usamos computadores, podemos fazer muitas coisas. Uma criança pode usar a Internet para passar uma mensagem, um estudante pode usar uma planilha eletrônica para calcular uma média ou quantos pontos precisa para ser aprovado em cada matéria, um cozinheiro pode guardar suas receitas em software como o Word ou em um produto especializado para receitas. Na verdade, a quantidade de produtos especializados é tão grande que, se você procurar bem, certamente vai encontrar algum programa que faça algo bem próximo do que você deseja.

O problema é que, às vezes, queremos fazer algo específico: queremos um programa de computador que faça algo que servirá de forma única a nós ou a nossa empresa. Nesse caso, em vez de comprar um programa pronto temos que desenvolver o nosso próprio programa. Para isso é necessário dominar uma nova forma de manipular o computador: a programação. Nosso motivo pode ser um negócio, um trabalho escolar, um hobby ou mera curiosidade. Hoje em dia, programar um computador pode ser feito de várias formas. Você pode, por exemplo, modificar levemente o comportamento de aplicações por meio de macros, como é permitido em programas como Microsoft Word. Você pode fazer ainda modificações mais sérias por meio de linguagens embutidas, como pode ser feito também nos programas do Microsoft Office ou até mesmo em jogos de computador como Neverwinter Nights. Você pode também pegar um programa já existente de código aberto, ou software livre e modificá-lo. Ou você pode começar do início e programar praticamente tudo, certamente com ajuda de bibliotecas prontas que fazem parte do trabalho.

Para programar você tem muitas opções: pacotes que podem ser estendidos com macros ou linguagens embutidas, ambientes de programação point-and-click, linguagens mais fáceis de aprender e linguagens mais difíceis, mas que

apresentam grande poder ou características apropriadas para grandes sistemas. Em todo caso, o espírito por trás de tudo é o mesmo: programar é dar ordens para o computador, mostrar como ele deve reagir ao usuário e como ele deve processar os dados disponíveis.

Praticamente não há limite do que você pode fazer com um computador. Computadores ajudam pessoas a falar, controlam aparelhos e levaram o homem a Lua de várias maneiras. Mesmo as coisas mais difíceis, como simular um sentimento ou inteligência, são estudadas com afinco em todo mundo. Alguns problemas são muito grandes e exigem a construção de computadores enormes. Outros são tão simples que podemos resolver em computadores simples, que estão dentro de equipamentos. A noção de poderoso também muda com o tempo: um chip que era usado em computadores pessoais em 1988, o w:Z80, hoje é usado em aparelhos como faxes.

Hoje é difícil imaginar um domínio da atividade humana onde a utilização de computadores não seja desejável. Assim sendo o domínio da programação é substancialmente ditado pela imaginação e criatividade. Podemos dizer que a grande vantagem de saber programar é a possibilidade de criar o que quiser, quando quiser. Não só para o PC, mas celulares, PDAs, entre outros. Claro que exige um pouco de esforço, porém para muitos esse esforço é na verdade um desafio cuja a recompensa é ver sua ideia transformada em realidade.

A programação

Provavelmente você já ouviu a palavra **programação**, conhece o seu significado, mas, provavelmente, desconhece o que faz, como se faz e quem faz. Programar é fácil e divertido, a dificuldade, para maioria dos iniciantes pouco persistentes, é começar a perceber como um computador funciona.

Bem, um computador pode ser entendido de várias maneiras. Dentro dele, o que existe são sinais eletrônicos. Os humanos que os projetos normalmente pensam nesses sinais como "1"s e "0"s. Em certo ponto, passamos a pensar em algo conhecido como linguagem de máquina, ou seja, sequências de "1"s e "0"s, normalmente escritos como números inteiros, que indicam um certo comportamento, como somar dois números. Para ficar mais fácil ainda, essa linguagem de máquina é normalmente transcrita para uma linguagem de montagem ou Assembly que descreve as ações que um computador pode fazer por meio de w:mnemônicos, como ADD e MOV. Porém, já há algum tempo, nós fazemos o computador funcionar por meio de programas escritos em linguagens de programação, que tentam deixar a tarefa de explicar o que o computador tem que fazer mais fácil para os seres humanos, mesmo que, por causa da alta especialização da linguagem, apenas a alguns deles. Todas as linguagens de programação têm essencialmente o mesmo propósito, que é permitir ao programador humano dar instruções ao computador.

No nosso cotidiano a comunicação é feita de um modo natural e raramente temos consciência das regras que aplicamos na nossa linguagem. O destino da aprendizagem de uma linguagem de programação é exatamente a mesma: aplicação de regras, se possível de forma tão arraigada que pareça ser inconsciente (abstrair). Um bom programador entende os "meandros" da linguagem que usa e pode até mesmo ver beleza, ou feiura, em um código, da mesma forma que gostamos ou não de um texto as vezes não pelo seu conteúdo, mas pela forma como foi escrito.

As linguagens são criadas com duas motivações: linguagens de uso geral, que servem para fazer "qualquer coisa" e linguagens de uso específico. Se você quer fazer programas que tratem de problemas estatísticos, provavelmente linguagens como "R", uma linguagem criada para esse uso específico, são adequadas. Se você quer fazer um programa para calcular a folha de pagamento de uma empresa, provavelmente linguagens como COBOL, C, C++ ou Java, linguagens de uso geral, serão adequadas.

Um Programa de Computador

Um programa de computador é como uma receita de cozinha: uma sequência de passos a serem executados. Se computadores cozinhassem em vez de processar dados, um programa típico poderia ser:

```
PROGRAMA FRITAR_OVO
  RESERVAR OVO, FRIGIDEIRA, SAL, MANTEIGA ;
  USAR FOGÃO;
  COLOCAR FRIGIDEIRA NO FOGÃO;
  COLOCAR MANTEIGA NA FRIGIDEIRA;
  LIGAR FOGÃO;
  ESPERAR MANTEIGA FICAR QUENTE;
  QUEBRAR OVO;
  DERRAMAR OVO NA FRIGIDEIRA;
  COLOCAR SAL NO OVO;
  ESPERAR OVO FICAR FRITO;
  DESLIGAR FOGÃO;
  SERVIR OVO;
END PROGRAMA
```

Porém, programas de computador trabalham com dados, e um programa típico real é (usando a linguagem Python)

```
def somar(num1, num2):
    return num1 + num2
```

Este programa (ou melhor, essa função) retorna a soma de dois números.

Como programar

Estrutura interna de um computador

Um computador minimalista é constituído por três unidades básicas:

- Processador, como o nome indica, é o componente principal do processamento;
- Memória, quem mantém dados e programas;
- Dispositivos de entrada e saída (*Input/Output*), tais como teclado, monitor ou impressora.

Em um computador pessoal, esses componentes normalmente estão colocados em uma placa mãe.

É importante notar que os chamados dispositivos de memória secundária se comunicam com a parte principal do computador por dispositivos de entrada e saída. Assim, um disco rígido só pode ser usado se conectado a placa mãe por meio de uma interface (SCSI ou SATA, por exemplo).

Usualmente, representamos um computador de forma abstrata por um diagrama muito simples que mostra uma unidade de processamento capaz de usar dados que provêm ou devem ser guardados tanto na memória quanto em dispositivos de entrada e saída:

Figura 1: Esquema genérico de um computador

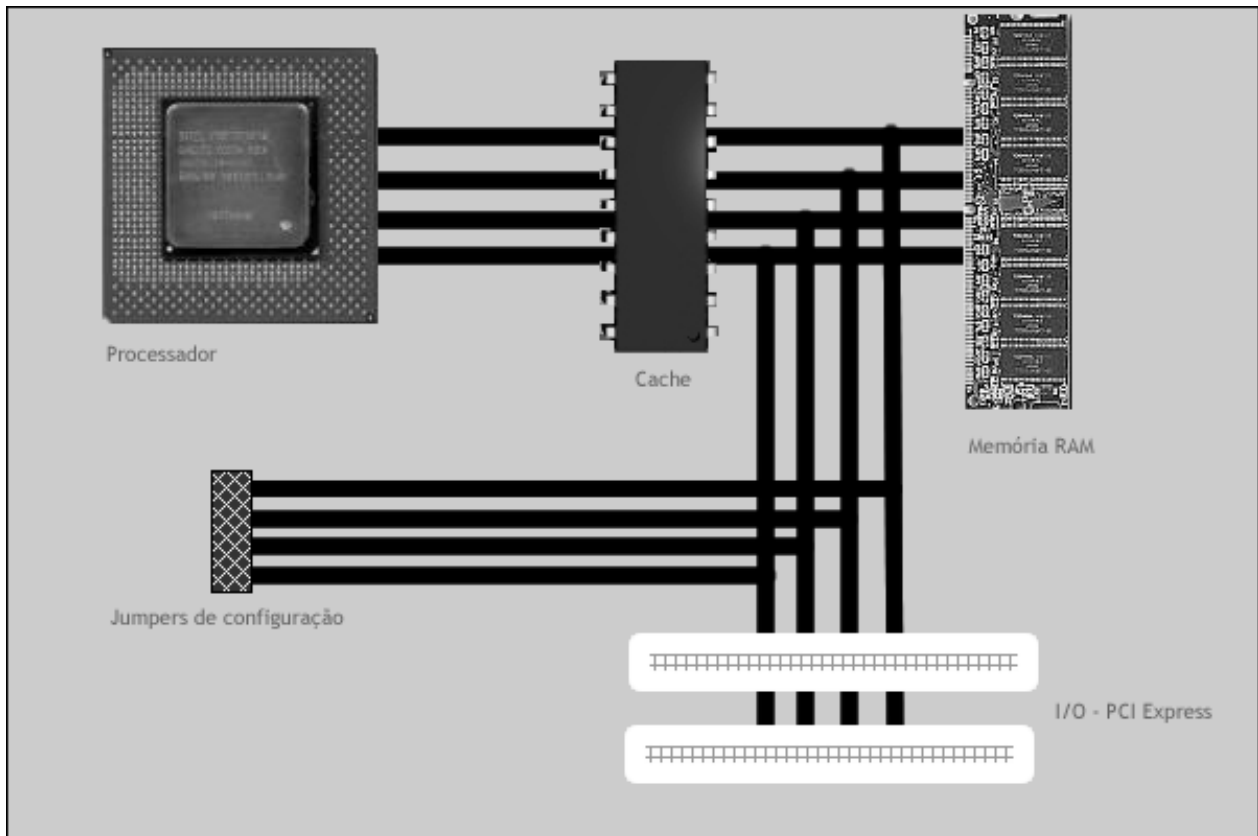


Figura 2: Esquema genérico de uma placa mãe

Antes de ficar perplexo a tentar perceber que esquema é aquele ali em cima, irei explicá-lo para o leitor compreender como um computador funciona no fundo.

O esquema apresenta dois dispositivos de entrada (PCI Express - aquelas onde nós colocamos a nossa placa gráfica, placa de rede ou placa de som...), quatro pistas de encaminhamento de dados (são mais, muitas mais num computador actual), onde circulam os dados, provavelmente codificados, provenientes das entradas, directas à central de processamento (CPU ou Processador). Aí, os milhões de transístores existentes dentro dessa caixinha, irão processar e criar novos dados que serão distribuídos pela rede interna do pc, segundo a codificação apresentada nos dados de entrada. O Processador pode guardar dados dentro da memória RAM e na memória Cache, sendo que, para a memória RAM irão dados menos usados e para a Cache os dados mais acessados pelo processador. Os Jumpers controlam, além da velocidade de processamento, que tipo de entradas poderão gerar dados, entre outras coisas. O mesmo processo se sucede aos dados que retornam aos dispositivos I/O. *Et voilà*, aqui está uma explicação muito, muito resumidinha de toda a teoria de processamento de um computador.

Alargando um pouco mais a escala, dispositivos periféricos, tais como impressoras e *scanners*, acessam também ao processador. Actualmente os dispositivos não são controlados pelo processador, cabendo isso a uma memória EEPROM chamada BIOS.

Se quiser conhecer mais sobre este assunto sugiro que procure pelos excelentes tutoriais da Guia do Hardware ^[1] sobre o tema (que, aliás, poderá ser uma casa interessante para quem quer aprender mais sobre hardware e Linux).

Processamento de dados

O processador é a unidade central do computador, designado por CPU (Central Processing Unit). A sua função é a de interpretar e executar instruções.

A unidade de medida da velocidade de um processador é o Hz (hertz). O hertz é a unidade de medida da frequência, que, em física, é definida como sendo o número de ciclos que ocorrem por unidade de tempo - a frequência de um relógio é 1/3600 Hz, ou seja, demora 1 hora até se dar uma volta completa. Nos computadores mais atuais, a velocidade média é de 1 Gigahertz, ou 1 bilhão de ciclos de relógio por segundo, ou 1 bilhão de hertz, ou ainda, analogamente, 1 bilhão de voltas completas no relógio em 1 segundo. No nosso exemplo, 01 hertz pode transportar no mínimo 01 bit (1 informação), para efeito de comparação 1 bit (1 hertz) pode ser comparado a 1 letra deste texto, logo computadores que trabalham com 2 bilhões de "letras" por segundo (02 Gigahertz) podem ler um livro mais rápido que outro que os que somente leem 1 bilhão de "letras" (01 Gigahertz).

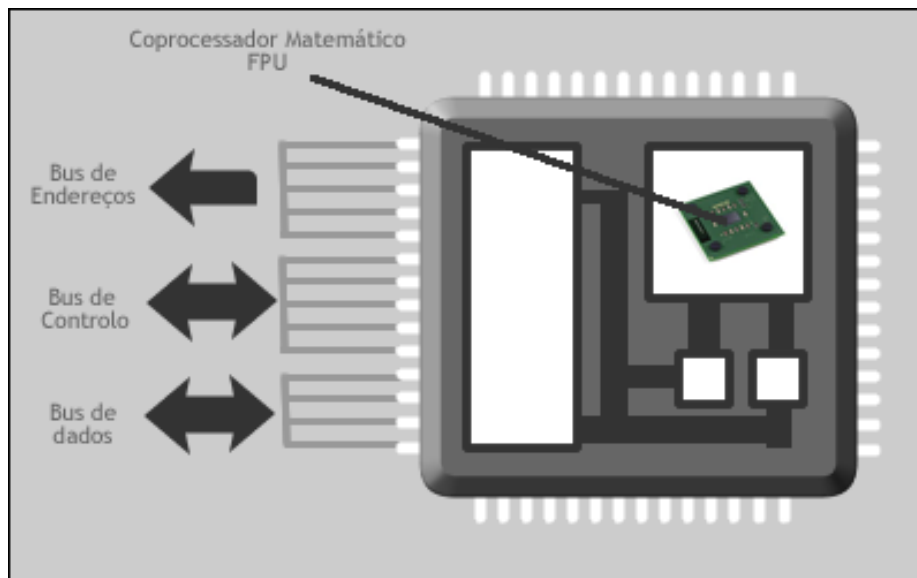


Figura 2 - Esquema genérico de um processador

O Processador é formado por milhões de transístores, onde cada um processa um bit de cada vez, ou seja, apresenta ou o estado 1 ou o estado 0. Esta diversidade de sequências possíveis cria um leque infinito de instruções. De facto as limitações encontradas aquando da criação de software não são encaradas pelo processador mas sim pela estrutura da máquina. O Processador, teoricamente, em termos de processamento de dados é ilimitado, não existe limites de processamento.

Por vezes são necessárias várias operações matemáticas complexas. Existe, dentro do próprio processador, uma pequena secção chamada Coprocessador Matemático FPU encarregada disso. Mas o processador não pode existir isoladamente, logo precisa de estar ligado por "algo": os Barramentos BUS do processador são os "caminhos" por onde a informação é encaminhada aos dispositivos do computador e vice-versa. Quanto maior o número de Bus mais rapidamente se dão as transferências. Existem várias tecnologias e protocolos usados no BUS. Siga o *link* BUS para saber mais sobre isso.

Referências

[1] <http://www.guiadohardware.net>

Definições sobre Lógica de Programação

Lógica de Programação é a técnica de desenvolver algoritmos (sequências lógicas) para atingir determinados objetivos dentro de certas regras baseadas na Lógica matemática e em outras teorias básicas da Ciência da Computação e que depois são adaptados para a Linguagem de Programação utilizada pelo programador para construir seu software.

Um algoritmo é uma sequência não ambígua de instruções que é executada até que determinada condição se verifique. Mais especificamente, em matemática, constitui o conjunto de processos (e símbolos que os representam) para efectuar um cálculo.

O conceito de algoritmo é freqüentemente ilustrado pelo exemplo de uma receita, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer iterações) ou necessitar de decisões (tais como comparações ou lógica) até que a tarefa seja completada. Um algoritmo corretamente executado não irá resolver um problema se estiver implementado incorretamente ou se não for apropriado ao problema.

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 autômato ou mesmo por um ser humano. Diferentes algoritmos podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros. Tal diferença pode ser reflexo da complexidade computacional aplicada, que depende de estruturas de dados adequadas ao algoritmo. Por exemplo, um algoritmo para se vestir pode especificar que você vista primeiro as meias e os sapatos antes de vestir a calça enquanto outro algoritmo especifica que você deve primeiro vestir a calça e depois as meias e os sapatos. Fica claro que o primeiro algoritmo é mais difícil de executar que o segundo apesar de ambos levarem ao mesmo resultado.

O conceito de um algoritmo foi formalizado em 1936 pela Máquina de Turing de Alan Turing e pelo cálculo lambda de Alonzo Church, que formaram as primeiras fundações da Ciência da Computação.

Formalismo

Um programa de computador é essencialmente um algoritmo que diz ao computador os passos específicos e em que ordem eles devem ser executados, como por exemplo, os passos a serem tomados para calcular as notas que serão impressas nos boletins dos alunos de uma escola. Logo, o algoritmo pode ser considerado uma sequência de operações que podem ser simuladas por uma máquina de Turing completa.

Quando os procedimentos de um algoritmo envolvem o processamento de dados, a informação é lida de uma fonte de entrada, processada e retornada sob novo valor após processamento, o que geralmente é realizado com o auxílio de uma ou mais estruturas de dados.

Para qualquer processo computacional teórico, o algoritmo precisa estar rigorosamente definido, especificando a maneira que ele se comportará em todas as circunstâncias. A correteza do algoritmo pode ser provada matematicamente, bem como a quantidade assintótica de tempo e espaço (complexidade) necessários para a sua execução. Estes aspectos dos algoritmos são alvo da análise de algoritmos. As implementações, porém, podem se limitar a casos específicos.

A maneira mais simples de se pensar um algoritmo é por uma lista de procedimentos bem definida, no qual as instruções são executadas passo a passo a partir do começo da lista, uma idéia que é pode ser facilmente visualizada através de um fluxograma. Tal formalização adota as premissas da programação imperativa, que é uma forma mecânica para visualizar e desenvolver um algoritmo. Concepções alternativas para algoritmos variam em programação funcional e programação lógica.

Término do algoritmo

Alguns autores restringem a definição de algoritmo para procedimentos que eventualmente terminam. Minsky constatou que se o tamanho de um procedimento não conhecido de antemão, tentar descobri-lo é problema indecidível já que o procedimento pode ser executado infinitamente, de forma que nunca se terá a resposta. Alan Turing provou em 1936 que não existe máquina de Turing para realizar tal análise para todos os casos, logo não há algoritmo para realizar tal tarefa para todos os casos. Tal condição é conhecida atualmente como problema da parada. Basicamente, isto quer dizer que não existe um programa de computador que possa antever, de forma geral, se um outro programa de computador vai parar algum dia.

Para algoritmos intermináveis o sucesso não pode ser determinado pela interpretação da resposta e sim por condições impostas pelo próprio desenvolvedor do algoritmo durante sua execução. Por exemplo, podemos querer um algoritmo interminável para controlar um sinal de trânsito.

Implementação

A maioria dos algoritmos é desenvolvida para ser implementada em um programa de computador. Apesar disso eles também podem ser implementados por outros modos tais como uma rede neural biológica (tal como no cérebro quando efetuamos operações aritméticas) em circuitos elétricos ou até mesmo em dispositivos mecânicos.

Para programas de computador existem uma grande variedade de linguagens de programação, cada uma com características específicas que podem facilitar a implementação de determinados algoritmos ou atender a propósitos mais gerais.

Análise de algoritmos

A análise de algoritmos é um ramo da ciência da computação que estuda as técnicas de projeto de algoritmos e os algoritmos de forma abstrata, sem estarem implementados em uma linguagem de programação em particular ou implementadas de algum outro modo. Ela preocupa-se com os recursos necessários para a execução do algoritmo tais como o tempo de execução e o espaço de armazenamento de dados. Deve-se perceber que para um dado algoritmo pode-se ter diferentes quantidades de recursos alocados de acordo com os parâmetros passados na entrada. Por exemplo, se definirmos que o fatorial de um número natural é igual ao fatorial de seu antecessor multiplicado pelo próprio número, fica claro que a execução de fatorial(10) consome mais tempo que a execução de fatorial(5).

Um meio de exibir um algoritmo afim de analisá-lo é através da implementação por pseudocódigo em português estruturado. O exemplo a seguir é um algoritmo em português estruturado que retorna (valor de saída) a soma de dois valores (também conhecidos como parâmetros ou argumentos, valores de entrada) que são introduzidos na chamada da função:

```
função SomaDeDoisValores (A numérico, B numérico)
início
    declare SOMA numérico
    SOMA <-- A + B
    retorne (SOMA)
fim
```

Classificação

Classificação por implementação

Pode-se classificar algoritmos pela maneira pelo qual foram implementados.

- **Recursivo ou iterativo** - um algoritmo recursivo possui a característica de invocar a si mesmo repetidamente até que certa condição seja satisfeita e ele é terminado, que é um método comum em programação funcional. Algoritmos iterativo usam estruturas de repetição tais como laços, ou ainda estruturas de dados adicionais tais como pilhas, para resolver problemas. Cada algoritmo recursivo possui um algoritmo iterativo equivalente e vice versa, mas que pode ter mais ou menos complexidade em sua construção. É possível construir algoritmos que sejam ao mesmo tempo iterativo e recursivo, provavelmente para aproveitar alguma otimização de tempo ou espaço que isso permita.
- **Lógico** - um algoritmo pode ser visto como uma dedução lógica controlada. O componente lógico expressa os axiomas usados na computação e o componente de controle determina a maneira como a dedução é aplicada aos axiomas. Tal conceito é base para a programação lógica.
- **Serial ou paralelo** - algoritmos são geralmente assumidos por serem executados instrução à instrução individualmente, como uma lista de execução, o que constitui um algoritmo serial. Tal conceito é base para a programação imperativa. Por outro lado existem algoritmos executados paralelamente, que levam em conta arquiteturas de computadores com mais de um processador para executar mais de uma instrução ao mesmo tempo. Tais algoritmos dividem os problemas em sub-problemas e o delegam a quantos processadores estiverem disponíveis, agrupando no final o resultado dos sub-problemas em um resultado final ao algoritmo. Tal conceito é base para a programação paralela. De forma geral, algoritmos iterativos são paralelizáveis; por outro lado existem algoritmos que não são paralelizáveis, chamados então problemas inerentemente seriais.
- **Determinístico ou não-determinístico** - algoritmos determinísticos resolvem o problema com uma decisão exata a cada passo enquanto algoritmos não-determinísticos resolvem o problema ao deduzir os melhores passos através de estimativas sob forma de heurísticas.
- **Exato ou aproximado** - enquanto alguns algoritmos encontram uma resposta exata, algoritmos de aproximação procuram uma resposta próxima a verdadeira solução, seja através de estratégia determinística ou aleatória. Possuem aplicações práticas sobretudo para problemas muito complexos, do qual uma resposta correta é inviável devido à sua complexidade computacional.

Classificação por paradigma

Pode-se classificar algoritmos pela metodologia ou paradigma de seu desenvolvimento, tais como:

Divisão e conquista - algoritmos de divisão e conquista reduzem repetidamente o problema em sub-problemas, geralmente de forma recursiva, até que o sub-problema é pequeno o suficiente para ser resolvido. Um exemplo prático é o algoritmo de ordenação merge sort. Uma variante dessa metodologia é o decremento e conquista, que resolve um sub-problema e utiliza a solução para resolver um problema maior. Um exemplo prático é o algoritmo para pesquisa binária. **Programação dinâmica** - pode-se utilizar a programação dinâmica para evitar o re-cálculo de solução já resolvidas anteriormente. **Algoritmo ganancioso** - um algoritmo ganancioso é similar à programação dinâmica, mas difere na medida que as soluções dos sub-problemas não precisam ser conhecidas a cada passo, uma escolha gananciosa pode ser feita a cada momento com o que até então parece ser mais adequado. **Programação linear** **Redução** - a redução resolve o problema ao transformá-lo em outro problema. É chamado também transformação e conquista. **Busca e enumeração** - vários problemas podem ser modelados através de grafos. Um algoritmo de exploração de grafo pode ser usado para caminhar pela estrutura e retornam informações úteis para a resolução do problema. Esta categoria inclui algoritmos de busca e backtracking. **Paradigma heurístico e probabilístico** - algoritmos probabilísticos realizam escolhas aleatoriamente. Algoritmos genéticos tentam encontrar

a solução através de ciclos de mutações evolucionárias entre gerações de passos, tendendo para a solução exata do problema. Algoritmos heurísticos encontram uma solução aproximada para o problema.

Classificação por campo de estudo

Cada campo da ciência possui seus próprios problemas e respectivos algoritmos adequados para resolvê-los. Exemplos clássicos são algoritmos de busca, de ordenação, de análise numérica, de teoria de grafos, de manipulação de cadeias de texto, de geometria computacional, de análise combinatória, de aprendizagem de máquina, de criptografia, de compressão de dados e de interpretação de texto.

Classificação por complexidade

Ver artigo principal: Complexidade computacional. Alguns algoritmos são executados em tempo linear, de acordo com a entrada, enquanto outros são executados em tempo exponencial ou até mesmo nunca terminam de serem executados. Alguns problemas possuem múltiplos algoritmos enquanto outros não possuem algoritmos para resolução.

Para lembrar

- Definição de Ciências da Computação

História da Programação

A mais antiga programadora de computadores de que se tem notícia é Ada Lovelace, que descreveu o funcionamento da máquina analítica de Charles Babbage, que nunca ficou pronta. O primeiro programador que completou todos os passos para a computação, incluindo a compilação e o teste, foi Wallace Eckert. Ele usou linguagem matemática para resolver problemas astronômicos na década de 1930. Alan Turing elaborou e programou um computador destinado a quebrar o código alemão ENIGMA na Segunda Guerra Mundial.

Lógica

Lógica binária

A **lógica binária**, ou *bitwise operation* é a base de todo o cálculo computacional. Na verdade, são estas operações mais básicas que constituem todo o poderio dos computadores. Qualquer operação, por mais complexa que pareça, é traduzida internamente pelo processador para estas operações.

Operações

NOT

O operador unário NOT, ou **negação binária** resulta no complemento do operando, i.e., será um bit '1' se o operando for '0', e será '0' caso contrário, conforme podemos confirmar pela tabela de verdade:

A	$\neg A$
1	0
0	1

Implementação:

```
Se isto NOT aquilo
```

AND

O operador binário AND, ou **conjunção binária** devolve um bit 1 sempre que **ambos** operandos sejam '1', conforme podemos confirmar pela tabela de verdade:

A	B	$A \wedge B$
1	1	1
1	0	0
0	1	0
0	0	0

Implementação:

```
Se isto AND aquilo, Fazer assim
```

OR

O operador binário OR, ou **disjunção binária** devolve um bit 1 sempre que **pelo menos um** dos operandos seja '1', conforme podemos confirmar pela tabela de verdade:

A	B	A ∨ B
1	1	1
1	0	1
0	1	1
0	0	0

Implementação:

Se isto OR aquilo, Fazer assim

XOR

O operador binário XOR, ou **disjunção binária exclusiva** devolve um bit 1 sempre que **apenas um** dos operandos seja '1', conforme podemos confirmar pela tabela de verdade:

A	B	A ⊕ B
1	1	0
1	0	1
0	1	1
0	0	0

Implementação:

isto XOR aquilo, Fazer assim

Shift

O operador unário de *bit shifting*, ou **deslocamento bit-a-bit**, equivale à multiplicação ou divisão por 2 do operando que, ao contrário dos casos anteriores, é um grupo de bits, e consiste no deslocamento para a esquerda ou para a direita do grupo de bits. O bit inserido é sempre 0, e o bit eliminado pode ser opcionalmente utilizado (flag CF dos registos do processador).

```
( 101011(43) >> 1 ) =    010101[1]
( 101011(43) << 1 ) = [1]010110
```

Aprendizagem

```
1. includ <stdio.h>
```

```
int main (void){
```

```
float x,y,z; float xl, zl, f; scanf("%f\n%f\n%f",&x, &y, &z); xl=(x<1)?(2*x):(x*x);xl, yl, zl
```

"O padrão eficaz"

O "Padrão Eficaz" é algo que existe empiricamente num programador, quando este pretende estudar uma nova linguagem. O conceito está aqui apenas formalizado e denominado por **Padrão Eficaz**.

Mas o que é o Padrão Eficaz?

Quando se pretende estudar uma linguagem, não se pode estar à espera que lendo determinado livro ou freqüentando determinado curso ficaremos a sabê-la perfeitamente. A verdade é que não necessitamos do livro para aprender a programar bem e o curso, para alguns, é só uma perda de tempo. A grande maioria dos livros serve como um auxílio ao estudo e não como um suporte base do estudo. Assim, o que a maior parte dos programadores fazem para aprender uma nova linguagem é:

- 1ª - Estudar a sintaxe da linguagem através de um livro ou manual.
- 2ª - Entender as diferenças desta linguagem para outras que já saibam - Isto é muito importante!
- 3ª - Fazer algo que vai realmente torná-lo um bom programador dessa linguagem: Ler código já feito.
- 4ª - Começar a escrever os seus próprios programas.

É preciso ter em atenção estes passos que são fundamentais. Assim, se não souber o que é determinada função, sempre pode ir ao manual e procurar. Não se prenda ao "marranço" do livro porque isso não o leva a nada.

Este padrão é eficaz porque, para um iniciante, é possível aprender-se uma linguagem em pouco mais de 5 ou 6 meses. Para um programador experiente basta apenas uma a duas semanas para cessar a aprendizagem de uma nova linguagem.

Após a passagem por essa linguagem, inscreva-se numa das centenas de listas existentes na internet e aperfeiçoe os seus conhecimentos ajudando outros usuários a criar programas *open source*. Verá que se sentirá feliz por ter chegado tão longe! É necessário ter em mente que se seguir estas regras aprenderá a ser um bom programador.

Índice - Lógica | Linguagens de programação

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Algoritmos

Um algoritmo é um esquema de resolução de um problema. Pode ser implementado com qualquer sequência de valores ou objectos que tenham uma lógica infinita (por exemplo, a língua portuguesa, a linguagem Pascal, a linguagem C, uma sequência numérica, um conjunto de objectos tais como lápis e borracha), ou seja, qualquer coisa que possa fornecer uma sequência lógica. Em baixo podemos ver um algoritmo implementado num fluxograma, sobre o estado de uma lâmpada:

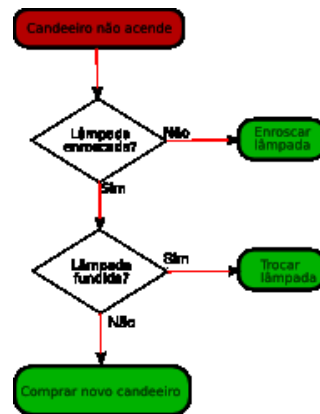


Figura 3 - Algoritmo num fluxograma

Seguindo o raciocínio em cima, então um programa de computador é já por si um algoritmo? Sim, é verdade. Embora tenhamos que usar um algoritmo prévio, na nossa língua (como apresentado na imagem acima) para escrever um programa com lógica, o próprio programa que provém desse algoritmo é já um algoritmo. Até um esquema mental é um algoritmo.

Ok, já percebi o que é um algoritmo. Mas porque é que isso interessa ao estudo da programação?

A verdade é que, antes de escrevermos um programa em qualquer outra linguagem é necessário escrever um esquema em papel para evitar erros, por exemplo, na nossa língua, segundo o programa que queremos fazer. Com isto não esquecemos a lógica que queremos dar ao programa e será menos comum o aparecimento de erros. Por exemplo:

Linguagem humana:

"Se for verdade isso, acontece isto, senão acontece aquilo"

Linguagem de máquina:

IF isso; THEN isto; ELSE aquilo;

O conteúdo escrito em cima está formalizado numa linguagem de algoritmia chamada *Portugol* ^[1] pela maior parte dos programadores e professores que trabalham em instituições de ensinamento das linguagens de programação. Como pode visualizar, um algoritmo pode ser escrito de várias maneiras, de cima para baixo, da esquerda para a direita, na diagonal, em árabe, em russo... É preciso é que o escreva!

Fundamentos

Uma máquina computacional é qualquer máquina (geralmente de origem eletro-eletrônica) com capacidade de receber dados, executar operações sobre estes dados e retornar os dados transformados por estas operações.

Entrada de Dados	Processamento	Saída de Dados
------------------	---------------	----------------

As máquinas computacionais eletro-eletrônicas possuem geralmente dois componentes básicos: software e hardware. Chamamos de **Hardware** sua parte física, e **software** os programas que tratam os dados imputados.

Quando inserimos algum dado em um computador, os dados inseridos são transformados em sinais elétricos (chamados de bits). O bit (do inglês *binary digit*) representa os dois estados (ligado ou desligado) que o sinal elétrico pode assumir. Para trabalhar com estes dados, podemos associar estes estados de ligado e desligado a 0 e 1. Quando utilizamos um computador, há um fluxo de sinais elétricos, que representam os dados inseridos, processados e retornados. Um conjunto de oito bits formam um *byte*, que é uma unidade completa de informação.

Dentro do byte, o estado de cada um dos oito bits, assim como sua posição relativa um ao outro, faz com que assumam o byte assumam um valor específico (não necessariamente numérico), que serve para estruturá-lo em relação a outros bytes e criar um sistema de dados que sirva ao usuário externo.

Para organizar as possibilidades de variações destes bits dentro de um byte, podemos visualizar uma tabela ASCII:

Binário	Dec	Hex	Representação
0010 0000	32	20	espaço ()
0010 0001	33	21	align="center"
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	\$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6

0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binário	Dec	Hex	Representação
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\

0101 1101	93	5D]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binário	Dec	Hex	Representação
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	
0111 1101	125	7D	}
0111 1110	126	7E	~

Lógica de Programação

Logicamente torna-se trabalhoso trabalhar com dados de computador bit-a-bit. Como forma de manipular este fluxo de estados elétricos e estruturá-lo de forma a permitir operações mais simplificadas e otimizadas sobre os bytes, surgiu o conceito de programação. As linguagens de programação são geralmente em dois níveis:

- **Linguagens de Baixo Nível:** são linguagens de programação que tratam a informação na linguagem de máquina.
- **Linguagens de Alto Nível:** são linguagens de programação modeladas quase como a linguagem comum humana, que quando compiladas são convertidas para linguagem de máquina. Cada linguagem deste tipo possui uma sintaxe própria, que deve ser respeitada e aprendida para que possa ser corretamente processada por seu compilador. Compilador é um programa que permite que determinada programação em uma linguagem específica seja adaptada para linguagem de máquina.

No entanto, não é necessário que o programador aprenda todas as diversas linguagens disponíveis no mercado. Cada linguagem é recomendada para determinadas aplicações, assim como possuem suas sintaxes próprias, mas todas são estruturadas logicamente. Com aprendizado da Lógica de Programação o aluno entenderá os conceitos básicos da programação poderá com menor ou maior dificuldade, dependendo da linguagem escolhida, aprender a linguagem que quiser.

Algoritmo

As linguagens de programação tratam os dados de um computador através do uso de algoritmos. Um algoritmo é uma estruturação passo-a-passo de como um determinado problema deve ser resolvido de forma não-ambigua (ou como muitos comparam "uma receita de bolo") . Desta forma, para realizar esta estruturação é necessário o uso de ferramentas e operações oriundas da Lógica, e principalmente da Lógica Matemática.

Antes de estruturar-se de forma lógica para programação, devemos saber qual o tipo de problema proposto, as informações que serão imputadas e os passos a serem efetuados para atingir-se um fim específico. Por exemplo, vamos ver um "algoritmo" sobre "tomar banho":

- 1.Tirar a roupa.
- 2.Abrir o registro.
- 3.Ensaboar-se.
- 4.Enxaguar o corpo.
- 5.Passar shampoo nos cabelos.
- 6.Enxaguar o cabelo.
- 7.Fechar o registro.

Vimos então um problema proposto (tomar banho) e os passos para solucionar o problema. Logicamente, que há outras formas de estruturarmos este algoritmo de forma a adaptá-lo a atingir o mesmo fim. No entanto, o importante é estruturá-lo de forma coerente, eficaz e simples, ou como muitos dizem de "forma elegante". Veremos na próxima lição que podemos desenhar este algoritmo e aplicar conectivos lógicos que permitam manipular as informações necessárias.

Exercícios

Para complementar os estudos baixe alguns exercícios de algoritmos ^[2]. Faça esses exercícios até sua lógica de programação ficar bem afiada.

Bibliografia

- Algoritmo, artigo na Wikipédia em português
- Bit, artigo na Wikipédia em português

Índice - Como programar | Lógica

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Referências

[1] <http://portugol.sites.uol.com.br>

[2] <http://algoritmizando.com/desenvolvimento/40-exercicios-de-algoritmos-resolvidos-para-estudo/>

Estruturas de manipulação de dados

Como já foi referido e é lógico, as linguagens de programação têm coisas em comum. Uma delas são as estruturas de controlo. Estruturas de Controlo são definidas como sendo a base da lógica da programação e podem ser de dois níveis: directo ou indirecto (complexo). Para termos uma ideia da diferença entre um controlo directo e um controlo indirecto, apresento a seguir dois diálogos representativos de duas situações do quotidiano:

Pedro – "Onde foste Miguel?"

Miguel – "Fui à loja comprar roupa."

Indivíduo – "Onde posso arranjar uma certidão A-R53?"

Inspector – "Tem que levar a sua identificação ao guiché, pedir um impresso GHA NORMAL carimbado para 3 meses. Após a conclusão da escritura terá que aguardar até que seja chamado pela sua vez."

Analisando os dois casos, rapidamente concluímos que a resposta obtida no primeiro é directa e bem mais simples que no segundo. Ora, se quisesse converter estas situações para uma situação parecida no computador, poderíamos constatar que bastava-nos uma página de um código sequenciado para o computador proceder a recriação do conteúdo da primeira situação. O mesmo já não se verifica no segundo. Normalmente para este tipo de casos, o programador utiliza peças fundamentais chamadas **funções** (do inglês **function**) que retratam cada um único acontecimento da situação, havendo assim ligações entre eles dependendo dos resultados obtidos. Funções são retratadas mais à frente. Penso que com estes dois exemplos o leitor já entende bem o que é um **Algoritmo directo** e um **Algoritmo complexo**.

Estruturas básicas

Qualquer programa tem que fazer alguma coisa (nem que seja enviar um sinal bip) senão não é um programa, ou seja, ele tem que apresentar conteúdo. Assim, como apresenta conteúdo, vai "alterar" estados dentro do computador, sendo que, o estado de uma das peças que vai inevitavelmente se alterar é a memória interna. É inevitável um programa não se "alojar" na memória do computador, assim, o programa precisa de um espaço físico na memória e para isso terá que o requisitar. É disso que iremos falar a seguir.

Variáveis e constantes

Uma variável é uma expressão que varia e normalmente é representado por uma incógnita **X**, e uma constante é uma expressão que não se altera (o número de Avogadro, o pi, o número de Neper) que pode ser representado por uma letra. Nos programas, variáveis são todas as expressões que podem ou não variar, assim como também podem ser constantes. Uma coisa é óbvia: Constantes não podem ser variáveis!

Na função seguinte Y assume o dobro de todos os valores de X:

$Y = 2X$ --> Nesta função, X varia por isso Y é uma variável

Na seguinte função Y assume sempre o valor 2:

$Y = 2$ --> Nesta função Y não varia e por isso é constante.

Aqui está patente a diferença de uma função variável e uma função constante em matemática.

Transportando a noção de Variável e Constante para a programação, apresento agora dois trechos de código em PHP e C++:

```
//Estou a representar um texto como uma variável representada por p
(sintaxe de PHP):
<?php
$p = "Olá Mundo!";
//Agora vou apresentá-la no ecrã
echo $p;
?>
```

No caso acima, uma variável é um texto que foi guardado em memória, representado por **p**.

```
//Represento agora um "Olá Mundo!" como uma constante (sintaxe do C++):
#include <iostream.h>;
int main() {
    cout << "Olá Mundo!";
    return 0;
}
```

Não se assuste se não entendeu nada do que se passou ali em cima. Aqui "Olá Mundo!" foi uma constante e não uma variável, pois ela não foi guardada em memória, mas imposta pela instrução **cout** directamente.

Instruções

As instruções são pequenos comandos que ditam ao programa o que fazer com determinado dado. Elas podem guardar informação, apresentar informação, aguardar um input, etc. Aqui apresentamos algumas instruções que são mais comuns assim como a sua implementação no **C++** e **PHP**:

Instrução	Descrição	PHP	Linguagem C++
MOSTRAR	Mostrar dados no ecrã	echo(); print();	cout <<
INPUT	Requisitar um input	\$_GET[] (Não é usado de forma directa)	cin.get()

Estruturas de controle

IF

IF é o mesmo que **SE** e é usado por todas as linguagens de programação por ser a estrutura mais simples que existe. A sua implantação representa o lançamento de um *booleano* **Verdadeiro** ou **Falso**.

```
IF isto
DO aquilo
```

ELSE

ELSE é usado como um acréscimo de **IF**, levando a que todos os dados retornados como **FALSO** em **IF** sejam controlados por este **ELSE**.

```
IF isto
DO aquilo
ELSE outra coisa
```

SWITCH

O **SWITCH** é visto como uma substituição de IF-ELSE quando existem mais de 2 opções a serem controladas.

```
SWITCH variável
CASE argumento 1: código correspondente
CASE argumento 2: código correspondente
CASE argumento 3: código correspondente
```

Neste caso, **SWITCH** irá procurar qual o argumento que variável contém e assim escolher qual dos CASE correr.

FOR

FOR é um loop que acontece enquanto determinado argumento não se tornar verdadeiro.

```
X = 1
FOR X <= 10
  X = X + 1
```

Pode ser mais difícil desvendar este código (principalmente se o leitor for matemático e ficar perplexo com a linha "X = X + 1"), mas o que está escrito ali em cima é que "enquanto X não for igual a 10, a instrução FOR vai sempre voltar ao início e somar/incrementar 1 ao valor de X".

A saída no processamento é a seguinte:

```
X = 1
X = 2
```

```
X = 3
X = 4
X = 5
X = 6
X = 7
X = 8
X = 9
X = 10
```

While

WHILE, estrutura utilizada em grande parte das linguagens de programação atuais, especifica uma ação enquanto determinada condição for verdadeira. Observe o exemplo escrito em pascal.

```
while (x <> z) do
  begin
    writeln ('Qual o valor de Z?')
    readln (z)
  end;
```

No código acima, enquanto o valor de Z for diferente do estabelecido para X, será pedido ao usuário que entre com o valor de Z.

Funções

Funções são pequenos trechos de código independentes que são especializados a tratar determinado tipo de dados dentro de um programa. Há linguagens, como o C ou o C++, que só trabalham com Funções, outras, como as linguagens de script (PHP, Python, Perl, etc) , que trabalham com funções e com código solto em sequência.

Exemplo de implementação de funções:

```
FUNCTION nome_da_função (argumento 1, argumento 2, argumento x,...)
  Código
RETURN dados a serem retornados ao código principal
```

Exemplo da implementação de funções num código:

```
FUNCTION multiplicador (numero)
  X = 10E21
  Y = numero * X
  RETURN Y
END-FUNCTION

GET numero
IF numero >= 1
  GOTO multiplicador
  SHOW Y
END-IF
ELSE
  SHOW "Não é número inteiro"
END-ELSE
```


Neste código ficámos a conhecer o poder das funções. No caso acima, é pedido ao utilizador a introdução de um número. Depois disso, o computador analisa se o número é inteiro, e se for, ele chama a função *multiplicador*, retornando uma variável **Y** contendo o número introduzido anteriormente multiplicado pelo expoente 22. Se pelo contrário, o número introduzido não for inteiro, o computador lança uma mensagem de erro "Não é número inteiro".

Arrays

Os Arrays são simples estruturas de dados, denominadas como *vector* ou *lista* se forem arrays uni-dimensionais, ou *matriz* se forem poli-dimensionais. O que acontece é que numa Array os dados são listados e ordenados segundo propriedades ou variáveis que tentamos dominar. No caso seguinte apresentamos uma array ordenada com dados sobre o estado de um programa open-source:

```
ARRAY ("versão" => ARRAY ("alpha" => 0.1
                          "beta"  => 0.5
                          "final" => 0.9
                        )
      "SO" => ARRAY ("win" => "Windows"
                    "uni" => "UNIX-Like"
                    "mac" => "Mac-OS"
                  )
      FIM-ARRAY;
    FIM-ARRAY;
```

Aos dados descriptos como "versão" e "SO" chamamos *Chave* e todos os outros são *Valores*. Então a todo um valor aponta uma chave.

```
ARRAY (Chave => Valor);
```

Operações aritméticas

Em qualquer linguagem, existe a possibilidade de se calcularem expressões algébricas aritméticas, segundo os sinais convencionais (+, -, *, e /), logo qualquer expressão que seja numérica se comporta como dada na matemática elementar. Porém, é possível usar-se expressões alpha-numéricas para se proceder a cálculos mais complexos (matéria em que não iremos entrar pois aqui as linguagens diferem no seu comportamento - algumas aceitam, outras calculam o valor hexadecimal ou ASCII do carácter, outras formam strings (frases), etc - levando assim a um leque infinito de possibilidades de programação, dependendo das necessidades previstas por cada tipo de linguagens).

Operações aritméticas

Podemos somar qualquer expressão algébrica do seguinte modo:

```
2 + 2 = A
MOSTRAR A
```

Obviamente que iríamos obter **4** como resultado mostrado. Analogamente, é possível calcular com qualquer calculador matemática básica:

```
4 * 5 = B
MOSTRAR B //resultado de 20

3 / 2 = C
```

```
MOSTRAR C //resultado de 0,(6)
```

Uma vez visto isto, poderíamos pensar que seria possível calcular expressões complexas

```
3 * 5 + 2 / 3 - 5 = D
MOSTRAR D
```

O que obtemos acima pode ser ambíguo, resultado da maneira como as linguagens de programação interpretam a expressão - calcular a expressão por sequência lógica matemática ou por sequência apresentada. Actualmente todas as linguagens mais comuns seguem a sequência lógica matemática para calcularem a expressão e assim o resultado obtido será **10,(6)** e não **0,(6)**.

Operações complexas

Como será que o computador se comporta com os cálculos que usam números de vírgula flutuante ou números exponenciais? Da mesma maneira como na matemática, o computador tratará de arredondar casas decimais (algumas linguagens como PHP necessitam da função **round()** para arredondar bem matematicamente) e de calcular potências, tudo isto para chegar a um resultado com um número real aproximado ao que se diz matematicamente verdadeiro.

```
Ponto flutuante:
0,512 + 2/3 = 1,178(6) //com quantas casas decimais o leitor quiser
e o computador permitir
```

Conclusão

Estas são as instruções mais básicas usadas por todas as linguagens de programação existentes, exactamente por serem básicas e simples. Sem elas, uma linguagem não era creditada e apresenta demasiados limites para ser considerada uma linguagem de programação racional.

Índice - Linguagens de programação | Orientação a objectos

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Pseudo-código

Pseudocódigo é uma forma genérica de escrever um algoritmo, utilizando uma linguagem simples (nativa a quem o escreve, de forma a ser entendido por qualquer pessoa) sem necessidade de conhecer a sintaxe de nenhuma linguagem de programação. É, como o nome indica, um pseudo-código e, portanto, não pode ser executado num sistema real (computador) — de outra forma deixaria de ser pseudo.

Os livros sobre a ciência de computação utilizam frequentemente o pseudocódigo para ilustrar os seus exemplos, de forma que todos os programadores possam entender as lógicas dos programas (independentemente da linguagem que utilizem). Entendendo-se os conceitos facilita-se depois a conversão para qualquer linguagem de programação. Aprenderemos neste curso de Lógica de Programação os elementos mais essenciais da programação utilizando o que conhecemos como *Português Estruturado*, isto é, um pseudo-código em português.

Constantes e variáveis

Uma máquina computacional é essencialmente uma máquina de entrada/saída de dados. Podemos definir dois tipos de dados : **constante**, que é um determinado valor fixo que não se altera até o término do programa e **variável** que corresponde a uma posição na memória do computador que armazena um determinado dado que pode ser modificado ao longo do programa.

Tipos de variáveis

Ao determinarmos uma variável ela aloca uma determinada posição na memória do computador. Assim há a necessidade de determinarmos o **tipo da variável** de modo a se ter espaço suficiente para a alocação de qualquer dado do tipo declarado.

- **Numérico:** variável que armazena dados de números. Alguns pseudo-códigos segmentam este tipo de variável em **real** e **inteiro**, i.e. dados numéricos reais (com casas decimais) e inteiros.
- **Caracter:** variável que armazena dados do jeito que são digitados- assim podem alocar letras, letras e números ou somente números, mas tratando estes números como texto e não como números em si mesmos.
- **Lógico:** variável que pode assumir apenas dois valores Verdadeiro ou Falso.

Modelo de pseudo-código

Utilizaremos como padrão o seguinte modelo de pseudo-código:

- Todo programa deve ser iniciado com **programa SeuNome**
 - O Início e Fim do programa serão limitados pelos marcadores **Início** e **Fim**
 - As variáveis serão declaradas no início do programa como NomeVariável:tipo da variável
 - Variáveis não podem ter espaço em branco e não podem ter seu nome iniciada por número
 - Caracteres especiais não devem ser utilizadas nos nomes das variáveis (‘,`, ~,ç , - e afins)
 - Deve-se evitar o uso de palavras-reservadas (i.e. aquelas utilizadas pelo programa para funções específicas como é o caso de Início e Fim até agora). Para facilitar ao estudante colocaremos estas palavras em **negrito**
 - Consideraremos que os nomes das variáveis são *case sensitive*, i.e. diferencia maiúsculas e minúsculas. Desta forma, o nome declarado de uma variável deve ser exatamente o mesmo, incluindo maiúsculas e minúsculas até o final.
 - Usaremos os comandos **leia** para receber dados do usuário e **escreva** para exibir dados ao usuário.
 - Os textos a serem exibidos na tela ou que tenham de ser inseridos como caracter serão colocados entre "aspas".
 - Os comentários sobre o código podem ser inseridos {entre chaves} servindo apenas para efeito de informação, mas não alterando o código.
-

Exemplo de programa em pseudo-código

Vamos agora criar um programa em pseudo-código que defina os tipos de variáveis relacionadas ao cadastro de um livro e receba estes dados do usuário e imprima na tela.

programa Livro {definição do nome do programa}

Início

CODIGODOLIVRO:**inteiro**

TITULO, AUTOR, EDITORA:**caracter** {declaração de variáveis}

escreva "Este é um programa em pseudo-código que exibe na tela os dados de um livro"

escreva "Digite o código do livro"

leia CODIGODOLIVRO

escreva "Digite o título do livro"

leia TITULO

escreva "Digite o autor do livro"

leia AUTOR

escreva "Digite a editora do livro"

leia EDITORA

escreva "O código do livro é", CODIGODOLIVRO

escreva "O título do livro é", TITULO

escreva "O autor do livro é", AUTOR

escreva "A Editora do livro é", EDITORA

Fim

Atribuição de valores às variáveis

As variáveis recebem valores do mesmo tipo de sua declaração no processamento do programa. No exemplo anterior associamos o valor digitado pelo usuário às variáveis. Se quisermos associar valores podemos utilizar <- que associa um valor a um identificador.

programa Livro {definição do nome do programa}

Início

CODIGODOLIVRO:**inteiro**

TITULO, AUTOR, EDITORA:**caracter** {declaração de variáveis}

escreva "Este é um programa em pseudo-código que exibe na tela os dados de um livro"

CODIGODOLIVRO <- 1

TITULO <- "O Senhor dos Anéis"

AUTOR <- "J.R.R.Tolkien"

EDITORA <- " Editora Tralalá"

escreva "O código do livro é", CODIGODOLIVRO {irá exibir 1}

escreva "O título do livro é", TITULO {irá exibir O Senhor dos Anéis}

escreva "O autor do livro é", AUTOR {irá exibir J.R.R.Tolkien}

escreva "A Editora do livro é", EDITORA {irá exibir Editora Tralalá}

Fim

Exercícios

Programa: maior de idade

Var: inteiro, nome

```
início
```

escreva: ("informe idade") leia ("idade") se idade <= 18 anos escreva:("pode tirar a carteira") senão ("não pode tirar a carteira")

```
fim se
```

```
fim
```

Expressões em pseudo-código

Introdução

Aqui serão descritas as funções e comandos usados no pseudocódigo.

Comandos iniciais

Esses comandos sempre estarão no pseudocódigo para fins de organização e não possuem nenhuma execução atribuída, ficam na seguinte ordem estrutural:

```
algoritmo "nome do algoritmo" {onde o que há em "" é uma variável literal}  
var  
{Seção de declaração de variáveis}  
início  
{Seção de início de comandos}  
finalgoritmo {Indicação fim do algoritmo}
```

Comando algoritmo

Só serve para indicar o nome do algoritmo, onde o nome deve vir entre aspas por ser uma variável literal, é obrigatório. Ex.:

```
algoritmo "teste"
```

Comando var

Indica onde as variáveis serão declaradas, é opcional, pois alguns algoritmos apenas imprimem instruções. Ex.:

```
var  
n1, n2      : inteiro  
n3, n4      : real  
nome, cep   : literal
```

Comando inicio

Indica onde começarão as instruções, é obrigatório. Ex.:

```
inicio  
Escreva("Isto é um algoritmo")
```

Comando finalgoritmo

Serve apenas para indicar que o algoritmo está terminado, é obrigatório.

Orientação a objectos

A **Orientação a Objeto** é um paradigma de Análise orientada ao objeto, Projeto orientado ao objeto e Linguagem de programação de sistemas de *software* baseado na composição e interação entre diversas unidades de software chamadas objetos.

Em alguns contextos, prefere-se usar Modelagem de dados orientada ao objeto, em vez de Projeto orientado ao objeto.

A análise e projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de *software*. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos.

Hoje existem duas vertentes no projeto de sistemas orientados a objetos. O projeto formal, normalmente utilizando técnicas como a notação UML e processos de desenvolvimento como o RUP; e a programação extrema, que utiliza pouca documentação, programação em pares e testes unitários.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de *software*. Cada classe determina o comportamento (definidos nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

Smalltalk, Perl, Python, Ruby, PHP, C++, Java e C# são as linguagens de programação mais importantes com suporte a orientação a objetos.

Conceitos

- **Classe** representa um conjunto de objetos com características afins. Uma classe define o comportamento dos objetos, através de métodos, e quais estados ele é capaz de manter, através de atributos.

Exemplo de classe:

```
HUMANO é uma classe e tem como atributos: 2 BRAÇOS, 2 PERNAS, 1 CABEÇA, etc...
```

- **Objeto** é uma instância de uma classe. Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.

Exemplo de objetos da classe Humanos:

JOÃO é um objecto da classe HUMANOS, apresentando todos os atributos dessa classe mas com a sua individualidade.

Objecto é portanto uma discriminação da classe, sendo a classe uma generalização de um conjunto de objectos idênticos ou com a mesma base.

- **Mensagem** é uma chamada a um objeto para invocar um de seus métodos, ativando um comportamento descrito por sua classe.
-

- **Herança** é o mecanismo pelo qual uma classe (subclasse) pode estender outra classe (superclasse), aproveitando seus comportamentos (métodos) e estados possíveis (atributos). Há herança múltipla quando uma subclasse possui mais de uma superclasse. Essa relação é normalmente chamada de relação "é um". Um exemplo de herança: Mamífero é superclasse de Humano. Ou seja, um Humano **é um** mamífero.
- **Associação** é o mecanismo pelo qual um objeto utiliza os recursos de outro. Pode tratar-se de uma associação simples "usa um" ou de um acoplamento "parte de". Por exemplo: Um humano usa um telefone. A tecla "1" é parte de um telefone.
- **Encapsulamento** consiste na separação de aspectos internos e externos de um objeto. Este mecanismo é utilizado amplamente para impedir o acesso direto ao estado de um objeto (seus atributos), disponibilizando externamente apenas os métodos que alteram estes estados. Exemplo:

Você não precisa conhecer os detalhes dos circuitos de um telefone para utilizá-lo. A carcaça do telefone encapsula esses detalhes, provendo a você uma interface mais amigável (os botões, o monofone e os sinais de tom).

- **Abstração** é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.
- **Polimorfismo** permite que uma referência de um tipo de uma superclasse tenha seu comportamento alterado de acordo com a instância da classe filha a ela associada. O polimorfismo permite a criação de superclasses abstratas, ou seja, com métodos definidos (declarados) e não implementados, onde a implementação ocorre somente nas subclasses não abstratas.

Índice - Estruturas de manipulação de dados | Exercícios

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Exercícios

Exercício 1

Verdadeiro ou falso

<quiz display=simple> {Não preciso de um outro programa aquando da finalização do meu código em C++. Basta-me rodá-lo directamente na máquina. ltype="()"} - Verdadeiro. + Falso.

{O Assembly é uma linguagem muito acessível ao usuário final. ltype="()"} - Verdadeiro. + Falso.

{Os *scripts* são linguagens dinâmicas e usadas em intervenções pequenas, para não se ter que usar o compilador. ltype="()"} + Verdadeiro. - Falso.

{Sempre preciso escrever um algoritmo antes de iniciar a escrita de um programa em uma linguagem. ltype="()"} - Verdadeiro. + Falso. </quiz>

Crie algoritmos

1. Crie um algoritmo próximo ao de máquina para a seguinte frase:

"Se estiveres no estado A tens que decifrar o código B e somar 2 ao resultado de B. Se não estiveres no estado A passa para o estado C e pára."

2. Decifre para o português o seguinte algoritmo:

```
IF Livro 1
GOTO Page 251
SOMA 2 NA Page = VAR
SAÍDA VAR
ELSE Livro 2
GOTO Page 23
SUBTRAI 2 NA Page = VAR
SAÍDA VAR
```

Exercício 2

Verdadeiro ou falso

<quiz display=simple> {Uma instrução FOR é uma instrução básica. ltype="()"} + Verdadeiro. - Falso.

{É exclusivamente necessário incluir-se a instrução ELSE numa IF. ltype="()"} - Verdadeiro. + Falso.

{SWITCH é bom para *loops*. ltype="()"} - Verdadeiro. + Falso.

{Variáveis guardam dados na memória do computador e podem ser manipuladas. ltype="()"} + Verdadeiro. - Falso. </quiz>

Crie algoritmos

1. Crie um algoritmo para esta situação da empresa "Pacheco-Car":

"O problema encontrado pelos nossos funcionários é que ao encomendar-se uma peça de automóvel, inserir-se a marca e requisitar o endereço, o programa não detecta se o registo termina em X ou em Y. É que se terminar em Y não podemos encomendar directamente. Temos que inserir o código de escape (que termina ou em 00 ou em 01 ou em 02) para o programa enviar a encomenda para os endereços A, B ou C, respectivamente.

2. Decifre o algoritmo apresentado em baixo:

```
Y = " polígono"
E = " não "
GET X
IF X=0
  MOSTRAR "Não posso aceitar o número!"
END-IF
ELSE
  SWITCH X
    CASE 1: "Eu sou" Y
    CASE 2: "Eu" E "sou" Y
    CASE 3: "Como podes aceitar" Y "aqui?"
  END-SWITCH
END-ELSE
```

Soluções dos exercícios

Índice - Orientação a objectos | Bibliografia

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Soluções dos exercícios

EXERCÍCIO 1

Verdadeiro ou falso

1. Falso | 2. Falso | 3. Verdadeiro | 4. Falso

Crie algoritmos

1.

```
IF A
  DECIFRAR B + 2 = VAR
ELSE
  C
STOP
```

2.

"Se tiveres o Livro 1, vai para a página 251, e mostra o valor da soma entre página e 2.
Se tiveres o Livro 2, vai para a página 23, e mostra o valor da subtração entre a página e 2."

Exercício 2

Verdadeiro ou falso

1. Verdadeiro | 2. Falso | 3. Falso | 4. Verdadeiro

Crie algoritmos

1.

```
GET REGISTO
IF REGISTO=Y
  GET ESCAPE
  SWITCH ESCAPE
    CASE "00": A
    CASE "01": B
    CASE "02": C
  END-SWITCH
END-IF
```

2.

```
"O programa vai pedir X e se este for igual a zero ele vai dizer que não pode aceitar esse número.
Se X for um ele vai-me responder que é polígono, se for igual a dois vai dizer que não é e se for
igual a três vai perguntar como poderia aceitar um polígono ali"
```

NOTA: Se acertou em tudo, parabéns! Pode-se considerar um perito na introdução às linguagens de programação. Se errou nalgum, sugiro que volte a estudar o capítulo necessário.

Voltar

Anexo: Linguagens de programação

História e evolução das linguagens

Este capítulo apresentará uma breve discussão sobre a história e evolução das linguagens de programação.

Assembly

Criada na década de 50, o Assembly foi das primeiras linguagens de programação a aparecer. Ela usa uma sintaxe complicada e "exageradamente" difícil, isto porque, antes da década de 50 os programadores de máquinas tinham que escrever instruções em código binário, qualquer coisa como: 0110010110011011010110011010111010110101 ... Para escrever uma instrução. Na verdade, o Assembly foi criado para facilitar o uso dessa tarefa, mas é considerado uma linguagem de baixo nível, pois tudo o que o processador interpreta tem que ser descrito pelo programador. Assim o código acima seria "add EAX" em Assembly. Bastava apenas, depois de estar concluída a escrita do código, rodar o compilador e tínhamos o programa.

- Vantagens: programas extremamente rápidos e pequenos.
- Desvantagens: tempo de desenvolvimento lento e sujeito a erros; código preso a uma arquitetura

Sobre Assembly:

- Assembly na Wikipédia

Fortran

Esta linguagem Fortran (Formula Translator) é uma linguagem de Alto nível, que foi criada partindo do problema e da dificuldade apresentadas pelo Assembly. Apareceu também na década de 50 e foi considerada uma das melhores linguagens da época. Aqui temos várias funções e instruções pré-definidas que nos permite poupar tempo na datilografia de instruções base do processador, ao contrário da linguagem Assembly.

- Vantagens:
- Desvantagens:

Sobre Fortran:

- Fortran na Wikipédia

Pascal

Outra linguagem de Alto nível dos anos 60, bem estruturada, mas com regras rígidas, o que a torna difícil de modelar, para se criar novas ideias. É a típica linguagem usada para iniciar os cursos de Programação. Actualmente ambientes de desenvolvimento (IDE) como o FreePascal, o Kylix e o Delphi são ótimas opções para se programar em Pascal.

- Vantagens: fortemente tipada (boa para iniciantes, os quais não tem muita familiaridade com a programação)
- Desvantagens: por ser fortemente tipada, prende programadores mais veteranos

Sobre Pascal:

- Pascal na Wikipédia
 - Pascal aqui na Biblioteca
-

Cobol

Foi uma linguagem usada para a criação e estruturação de bancos de dados financeiros nos anos 60 que ainda hoje é usada por este tipo de serviços. Comparada com o Pascal e o Assembly, esta linguagem é bem amigável e bastante acessível e actualmente serve para várias tarefas.

Sobre Cobol:

- Cobol na Wikipédia

Linguagem C

Poder-se-ia dizer que o C é uma das maravilhas das linguagens de programação. Muitos dos programas existentes hoje foram escritos nesta linguagem. O C foi desenvolvido nos laboratórios Bell na década de 70, e possui as seguintes características:

- Portabilidade entre máquinas e sistemas operacionais
- Dados compostos em forma estruturada
- Total interação tanto com o SO como com a máquina
- Código compacto e rápido.

Nos anos 80, C era a linguagem mais utilizada por programadores, por permitir a escrita intensiva de todas as características das linguagens anteriores. O Próprio UNIX e Linux foram escritos em C, assim como o front-end do MS-DOS, **Windows** e as aplicações Office mais usadas no mundo (OpenOffice.org, Microsoft Office, embora cada uma delas incluísse suas próprias linguagens de script), sendo também utilizada em aplicações gráficas e criação de efeitos especiais nos filmes Star Trek e Star Wars.

- Vantagens: programas extremamente rápidos e pequenos.
- Desvantagens: tempo de desenvolvimento lento e sujeito a erros

Sobre C:

- Wikilivro *Programar em C*
- C na Wikipédia
- Aprendendo a Linguagem C ^[1]

O C++

Uma linguagem que adiciona ao C um conjunto de recursos a mais, como o próprio nome sugere. O C++ é o C orientado a objetos. Avançando nos 90, passou por diversas atualizações e padronizações nesta época, o padrão do C++ ^[2] foi exaustivamente trabalhado pelos desenvolvedores durante oito anos, quando finalmente foi aprovado pelo ANSI. Vários projectos como o KDE (front-end para UNIX, Linux, BSD e recentemente para Windows) são escritos em C++.

- Vantagens: programas extremamente rápidos e pequenos; proteção contra alguns erros comuns em C
- Desvantagens: tempo de desenvolvimento lento

Sobre C++:

- Wikilivro *Programar em C++*
 - C++ na Wikipédia
-

Java, C#

As linguagens em ascensão no fim dos anos 90 e começo do ano 2000, são linguagens de alto poder de abstração e com boas capacidades de virtualização, o que lhes conferem boas possibilidades de independência de plataforma, embora estas características ainda estão sendo melhoradas.

- Vantagens: uma maior facilidade em C/C++ e vínculos de patentes com as empresas que as desenvolveram.

PHP

O PHP apareceu em 1994 e pretendeu revolucionar o mercado de linguagens na criação de *scripts* para a internet. Realmente é uma linguagem excepcional onde é permitido fazer tudo o que os CGIs faziam inclusive mais coisas ainda. Para quem quer seguir programação para aplicações web é uma linguagem a estudar, assim como o Perl, usado também na criação de ferramentas em *sites*.

- Link Para o *Curso de PHP* no Wikilivros
- Vantagens: facilidade de implementação e execução.
- Desvantagens: certa lentidão, que depende do ambiente onde foi instalado o servidor.

Perl, Python, Ruby

Ciclos de processamento e tempo de computador são cada vez mais baratos; tempo de programador e criatividade são cada vez mais caros. Por isso, a tendência atual no mercado é favorecer linguagens de alto nível, menos otimizadas para a máquina, e mais otimizadas para o programador: linguagens como Perl, Python e Ruby, consideradas linguagens de programação de alto nível, com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana.

- Linguagem *Python* desenvolvida no Wikilivros
- Vantagens: maior facilidade de implementação e execução em relação ao Java e ao C#
- Desvantagens: programas mais lentos do que em C/C++

Mais sobre o assunto

- Lista de linguagens de programação existentes
- Exemplos de Algoritmos em várias linguagens de programação
- O que são linguagens de programação

Igualdades e diferenças nas linguagens

As igualdades entre as várias linguagens são óbvias: A Lógica binárias, as instruções if, else, goto, switch, etc... Entre outros processos. Porém é necessário notar-se que a sintaxe básica dessas linguagens modifica-se e é necessária a sua aprendizagem. Para isso o melhor local para aprender e se aperfeiçoar são os livros sobre a matéria.

O problema mais importante, hoje, na área de Linguagens de Programação, é o desenvolvimento de linguagens que aumentem a produtividade do programador. Linguagens que permitam escrever programas corretamente, esta é a razão do rápido crescimento de linguagens como Java, C# e Ruby.

Por enquanto já pode encontrar Manuais de PHP, C++, C, Python e Javascript aqui na Wikibooks.

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Referências

[1] <http://www.global.estgp.pt/engenharia/Alunos/eSebentas/Tutoriais/c.htm>

[2] http://www.research.att.com/~bs/iso_release.html

Anexo: Passagem para a linguagem de programação escolhida

O objetivo deste livro e dos cursos de Ciências da Computação^[1] é ensinar os conceitos mais genéricos possíveis em todos os segmentos, e também em Programação. Isto significa que nenhuma linguagem será privilegiada em prejuízo de outra, já que entendemos que cada linguagem é apropriada para determinados tipos de aplicações e problemas. Nesta lição veremos no entanto algumas das principais linguagens, e como aplicar o conceito de Lógica de Programação a estas linguagens.

Referências

[1] Veja os cursos do Departamento de Computação e Informática Básica, na Wikiversidade

Bibliografia

Bibliografia

- Downey, Allen B.. *How To Think Like A Computer Scientist*^[1]: Learning with C++. Acesso em: 23 Julho 2005.
 - Walnum, Clayton. *O mais completo guia sobre princípios de programação*. São Paulo: Berkeley, 2002. 304 p. ISBN 8572516255
 - Hickson, Rosangela. *C++ Técnicas Avançadas*. 1ª.ed. Rio de Janeiro: Campus, 2003. 452 p. ISBN 8535212752
 - Morimoto, Carlos. *Guia do Hardware*^[1]. Acesso em 7 de Abril de 2006.
 - Programação de Sistemas^[2] - por Ivan Luiz Marques Ricarte
-

Ligações externas & mais sobre o assunto

- **Guia do Hardware** ^[1] - O mais completo portal com manuais, tutorias, referências, testes e notícias relacionadas com hardware e Linux. É também a actual casa do prezado Kurumin Linux.

Pratique

- Olimpíada Brasileira de Informática-Pratique ^[3]

Dúvidas, esclarecimentos e sugestões

- Envie um mail para: lightningspirit@gmail.com ^[4]

Índice - Exercícios - Bibliografia

Prefácio Programar, o que é a programação? Como programar: pensar como uma máquina Aprendizagem: o "padrão eficaz" Linguagens de programação Estruturas de manipulação de dados Orientação a objectos Exercícios Bibliografia e ligações externas

Referências

- [1] <http://www.ibiblio.org/obp/thinkCS/cpp/english/>
 - [2] <http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node1.html>
 - [3] <http://olimpiada.ic.unicamp.br/pratique>
 - [4] <mailto:lightningspirit@gmail.com>
-

Fontes e Editores da Página

Índice *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=227712> *Contribuidores:* Albmont, Alguém, Atoj, Dante Cardoso Pinto de Almeida, Helder.wiki, Jorge Morais, LeonardoG, Lightningspirit, Marcos Antônio Nunes de Moura, Master, Raylton P. Sousa, Rã, Voz da Verdade, Wbrito, Wikimi-dhiann, 7 edições anónimas

Prefácio *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=227713> *Contribuidores:* Albmont, Helder.wiki, Jorge Morais, Lightningspirit, Wbrito, 2 edições anónimas

Programar *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=232520> *Contribuidores:* Helder.wiki, Jorge Morais, João Jerónimo, LeonardoG, Lightningspirit, Marcos Antônio Nunes de Moura, Rã, Wbrito, Xexeo, 14 edições anónimas

Como programar *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=234295> *Contribuidores:* Atoj, Helder.wiki, Jorge Morais, Lightningspirit, Wbrito, Xexeo, 7 edições anónimas

Definições sobre Lógica de Programação *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=220140> *Contribuidores:* Hoo man, Jorge Morais, Master, Ozymandias, Xexeo, 7 edições anónimas

História da Programação *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204910> *Contribuidores:* Ricardo Augustinis, 1 edições anónimas

Lógica *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204912> *Contribuidores:* Dante Cardoso Pinto de Almeida, Jorge Morais, Lightningspirit, Marcos Antônio Nunes de Moura, Raylton P. Sousa

Aprendizagem *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204902> *Contribuidores:* Helder.wiki, Jorge Morais, Lightningspirit, Wbrito, 4 edições anónimas

Algoritmos *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=235489> *Contribuidores:* Albmont, Helder.wiki, Jorge Morais, MGFE Júnior, Master, Pedro Mendes, Wbrito, 12 edições anónimas

Estruturas de manipulação de dados *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=235425> *Contribuidores:* Atoj, Clauix, Eduobay, Helder.wiki, Hycesar, Jorge Morais, Lightningspirit, MGFE Júnior, Raylton P. Sousa, SallesNeto BR, 11 edições anónimas

Pseudo-código *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=228758> *Contribuidores:* Albmont, Jorge Morais, Master, Ozymandias, 7 edições anónimas

Expressões em pseudo-código *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=233405> *Contribuidores:* Albmont, Jorge Morais, Marcos Antônio Nunes de Moura, Master, Typeyotta, 9 edições anónimas

Orientação a objectos *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204913> *Contribuidores:* Albmont, Atoj, Helder.wiki, Jorge Morais, Lightningspirit, 1 edições anónimas

Exercícios *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204908> *Contribuidores:* Albmont, Helder.wiki, Jorge Morais, Lightningspirit, 2 edições anónimas

Soluções dos exercícios *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204918> *Contribuidores:* Atoj, Helder.wiki, Jorge Morais, Lightningspirit, 1 edições anónimas

Anexo: Linguagens de programação *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=234893> *Contribuidores:* Albmont, Atoj, Helder.wiki, Jocile, Jorge Morais, Lightningspirit, Marcos Antônio Nunes de Moura, Raylton P. Sousa, 16 edições anónimas

Anexo: Passagem para a linguagem de programação escolhida *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204914> *Contribuidores:* Helder.wiki, Jorge Morais, Master, Ozymandias

Bibliografia *Fonte:* <http://pt.wikibooks.org/w/index.php?oldid=204903> *Contribuidores:* Alguém, Helder.wiki, Jorge Morais, LeonardoG, Lightningspirit, Raylton P. Sousa, Rã, 2 edições anónimas

Fontes, Licenças e Editores da Imagem

Imagem:100%.svg Fonte: <http://pt.wikibooks.org/w/index.php?title=Ficheiro:100%.svg> Licença: Public Domain Contribuidores: Siebrand

Imagem:75%.svg Fonte: <http://pt.wikibooks.org/w/index.php?title=Ficheiro:75%.svg> Licença: Public Domain Contribuidores: Siebrand

Imagem:00%.svg Fonte: <http://pt.wikibooks.org/w/index.php?title=Ficheiro:00%.svg> Licença: Public Domain Contribuidores: Siebrand

Imagem:50%.svg Fonte: <http://pt.wikibooks.org/w/index.php?title=Ficheiro:50%.svg> Licença: Public Domain Contribuidores: Siebrand

Imagem:25%.svg Fonte: <http://pt.wikibooks.org/w/index.php?title=Ficheiro:25%.svg> Licença: Public Domain Contribuidores: Karl Wick

Imagem:Nuvola apps bookcase.png Fonte: http://pt.wikibooks.org/w/index.php?title=Ficheiro:Nuvola_apps_bookcase.png Licença: desconhecido Contribuidores: Acul99, Alno, Alphax, CyberSkull, Dbc334, Pluke, Pseudomoi, Rocket000, Shizhao, VIGNERON, 1 edições anónimas

Image:Seaman_send_Morse_code_signals.jpg Fonte: http://pt.wikibooks.org/w/index.php?title=Ficheiro:Seaman_send_Morse_code_signals.jpg Licença: Public Domain Contribuidores: Tucker M. Yates

Imagem:introducao_programacao_esquema_pc.png Fonte: http://pt.wikibooks.org/w/index.php?title=Ficheiro:Introducao_programacao_esquema_pc.png Licença: GNU Free Documentation License Contribuidores: Dante Cardoso Pinto de Almeida, Lightningspirit

Imagem:Introducao_programacao_esquema_processador.png Fonte: http://pt.wikibooks.org/w/index.php?title=Ficheiro:Introducao_programacao_esquema_processador.png Licença: GNU Free Documentation License Contribuidores: Dante Cardoso Pinto de Almeida, Lightningspirit

Imagem:Introducao_programacao_fluxograma.png Fonte: http://pt.wikibooks.org/w/index.php?title=Ficheiro:Introducao_programacao_fluxograma.png Licença: GNU Free Documentation License Contribuidores: Dante Cardoso Pinto de Almeida, Lightningspirit

Licença

Creative Commons Attribution-Share Alike 3.0 Unported
//creativecommons.org/licenses/by-sa/3.0/
