

Módulo 1

Introdução à Programação I



Lição 1

Introdução à Programação de Computadores

Autor

Florence Tiu Balagtas

Equipe

Joyce Avestro

Florence Balagtas

Rommel Feria

Reginald Hutcherson

Rebecca Ong

John Paul Petines

Sang Shin

Raghavan Srinivas

Matthew Thompson

Necessidades para os Exercícios**Sistemas Operacionais Suportados****NetBeans IDE 5.5** para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware**Nota:** IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em <http://java.sun.com/j2se/1.5.0/download.html>
- Para **Mac OS X**, Java 2 Platform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: <http://developer.apple.com/java> (é necessário registrar o download da JDK).

Para mais informações: <http://www.netbeans.org/community/releases/55/relnotes.html>

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori	Hugo Leonardo Malheiros Ferreira	Mauro Regis de Sousa Lima
Alexis da Rocha Silva	Ivan Nascimento Fonseca	Namor de Sá e Silva
Aline Sabbatini da Silva Alves	Jacqueline Susann Barbosa	Néres Chaves Rebouças
Allan Wojcik da Silva	Jader de Carvalho Belarmino	Nolyanne Peixoto Brasil Vieira
André Luiz Moreira	João Aurélio Telles da Rocha	Paulo Afonso Corrêa
Andro Márcio Correa Louredo	João Paulo Cirino Silva de Novais	Paulo José Lemos Costa
Antonie de Assis Lima	João Vianney Barrozo Costa	Paulo Oliveira Sampaio Reis
Antonio Jose R. Alves Ramos	José Augusto Martins Nieviadonski	Pedro Antonio Pereira Miranda
Aurélio Soares Neto	José Leonardo Borges de Melo	Pedro Henrique Pereira de Andrade
Bruno da Silva Bonfim	José Ricardo Carneiro	Renato Alves Félix
Bruno dos Santos Miranda	Kleberth Bezerra G. dos Santos	Renato Barbosa da Silva
Bruno Ferreira Rodrigues	Lafaiete de Sá Guimarães	Reyderson Magela dos Reis
Carlos Alberto Vitorino de Almeida	Leandro Silva de Moraes	Ricardo Ferreira Rodrigues
Carlos Alexandre de Sene	Leonardo Leopoldo do Nascimento	Ricardo Ulrich Bomfim
Carlos André Noronha de Sousa	Leonardo Pereira dos Santos	Robson de Oliveira Cunha
Carlos Eduardo Veras Neves	Leonardo Rangel de Melo Filardi	Rodrigo Pereira Machado
Cleber Ferreira de Sousa	Lucas Mauricio Castro e Martins	Rodrigo Rosa Miranda Corrêa
Cleyton Artur Soares Urani	Luciana Rocha de Oliveira	Rodrigo Vaez
Cristiano Borges Ferreira	Luís Carlos André	Ronie Dotzlaw
Cristiano de Siqueira Pires	Luís Octávio Jorge V. Lima	Rosely Moreira de Jesus
Derlon Vandri Aliendres	Luiz Fernandes de Oliveira Junior	Seire Pareja
Fabiano Eduardo de Oliveira	Luiz Victor de Andrade Lima	Sergio Pomerancblum
Fábio Bombonato	Manoel Cotts de Queiroz	Silvio Sznifer
Fernando Antonio Mota Trinta	Marcello Sandi Pinheiro	Suzana da Costa Oliveira
Flávio Alves Gomes	Marcelo Ortolan Pazzetto	Tásio Vasconcelos da Silveira
Francisco das Chagas	Marco Aurélio Martins Bessa	Thiago Magela Rodrigues Dias
Francisco Marcio da Silva	Marcos Vinicius de Toledo	Tiago Gimenez Ribeiro
Gilson Moreno Costa	Maria Carolina Ferreira da Silva	Vanderlei Carvalho Rodrigues Pinto
Givailson de Souza Neves	Massimiliano Girolidi	Vanessa dos Santos Almeida
Gustavo Henrique Castellano	Mauricio Azevedo Gamarra	Vastí Mendes da Silva Rocha
Hebert Julio Gonçalves de Paula	Mauricio da Silva Marinho	Wagner Eliezer Roncoletta
Heraldo Conceição Domingues	Mauro Cardoso Mortoni	

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos** - Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo** - Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani** - Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes** - Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso** - Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™

Rommel Faria – Criador da Iniciativa JEDI™

1. Objetivos

Nesta seção, vamos discutir os componentes básicos de um computador, tanto em relação a hardware como a software. Também veremos uma pequena introdução sobre linguagens de programação e sobre o ciclo de vida do desenvolvimento. Por fim, mostraremos os diferentes sistemas numéricos e as conversões entre eles.

Ao final desta lição, o estudante será capaz de:

- Identificar os diferentes componentes de um computador.
- Conhecer as linguagens de programação e suas categorias.
- Entender o ciclo de vida de desenvolvimento de programas e aplicá-lo na solução de problemas.
- Conhecer os diferentes sistemas numéricos e as conversões entre eles.

2. Introdução

O computador é uma máquina que realiza uma variedade de tarefas de acordo com instruções específicas. É uma máquina de processamento de dados, que recebe dados através de um **dispositivo de entrada** e o **processador** os manipula de acordo com um **programa**.

O computador tem dois componentes principais. O primeiro é o **Hardware** que é a parte palpável (que se pode tocar) do computador. Ele é composto de partes eletrônicas e mecânicas.

O segundo componente principal é o **Software** que é a parte impalpável (não se pode tocar) do computador. Ele é composto de dados e dos programas de computador.

3. Componentes Básicos de um Computador

3.1. Hardware

3.1.1. Unidade Central de Processamento (CPU - Central Processing Unit)

O processador é o “cérebro” do computador. Ele possui milhões de partes elétricas muito pequenas. Ele faz as operações fundamentais dentro do sistema. Alguns exemplos de processadores são o Pentium, Athlon e SPARC.

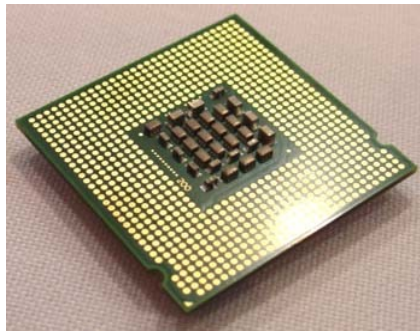


Figura 1: Processador de um PC

3.1.2. Memória

A memória, onde se encontram os dados e as instruções que a CPU precisa para realizar suas tarefas, dividida em diversos locais de armazenamento que possuem seus respectivos endereços lógicos. A CPU acessa a memória pelo uso destes endereços.

1. Memória Principal

A memória principal, às vezes, chamada de RAM (Random Access Memory ou Memória de Acesso Randômico) está fortemente ligada ao processador. Ela é utilizada para armazenar programas e dados, com os quais o processador está trabalhando no momento e não é utilizada para armazenamento de longo prazo, por este motivo seu armazenamento é considerado **volátil**. Isto significa que assim que o computador é desligado, toda a informação armazenada na memória principal será perdida.



Figura 2: Pente de memória

2. A Memória Secundária

A memória secundária está ligada à memória principal. Ela é usada para armazenar programas e dados para uso de longo prazo. Exemplos de memória secundária são discos rígidos e cd-rom.



Figura 3: Interior de um Disco rígido do computador

A memória secundária é considerada um tipo de armazenamento **não-volátil**. Isto significa que as informações nela armazenadas não serão perdidas após o computador ser desligado.

Memória Principal	Memória Secundária	Propriedade
Rápida	Lenta	Velocidade
Cara	Barata	Preço
Baixa	Alta	Capacidade
Sim	Não	Volátil

Tabela 1: Comparação entre a memória principal e a memória secundária

3.1.3. Dispositivos de Entrada e Saída

Os dispositivos de entrada e saída permitem que o computador interaja com o mundo exterior pela movimentação de dados para dentro e para fora do sistema.

Exemplos de dispositivos de entradas são teclados, mouses, microfones, etc. Exemplos de dispositivos de saída são monitores, impressoras, alto-falantes, etc.

3.2. Software

O software é um programa que o computador usa para funcionar. Ele é armazenado em algum dispositivo de hardware como um disco rígido, mas é em si mesmo intangível. Os dados que o computador usa podem ser qualquer coisa que o programa precise. Os programas agem como instruções para o processador.

Alguns tipos de programas de computador:

1. Programas de Sistemas

Programas necessários para que o hardware e o software funcionem juntos corretamente.

Exemplos:

- Sistemas Operacionais como Linux, Windows, Unix, Solaris, MacOS

2. Aplicativos

Programas que as pessoas usam para realizar determinado trabalho.

Exemplos:

- Processadores de Textos
- Jogos
- Planilhas Eletrônicas

3. Compiladores

O computador entende apenas uma linguagem: linguagem de máquina. Linguagem de máquina está na forma de zeros e uns. Já que é totalmente impraticável para as pessoas criarem programas usando zeros e uns, é preciso haver uma maneira de traduzir ou converter a linguagem que entendemos em linguagem de máquina, para isto, existem os compiladores.

4. Visão Geral sobre Linguagens de Programação

4.1. O que é uma linguagem de programação?

Uma linguagem de programação é uma técnica de comunicação padronizada para se expressar instruções para um computador. Assim como os idiomas utilizados pelos seres humanos, cada linguagem tem sua própria sintaxe e gramática.

Linguagens de programação possibilitam ao programador especificar precisamente com quais dados o computador irá interagir, como estes dados serão gravados/transmitidos, e precisamente quais ações serão tomadas de acordo com as circunstâncias.

Existem diferentes tipos de linguagens de programação que podem ser usadas para a criação de programas, mas, independente da linguagem utilizada, essas instruções são traduzidas em linguagem de máquina, e podem ser entendidas por computadores.

4.2. Categorias das Linguagens de Programação

1. Linguagens de Programação de Alto Nível

Uma linguagem de programação de alto nível é uma linguagem de programação que é mais amigável para o usuário, em alguns casos independente de plataforma, e que abstrai operações de baixo nível como acesso a memória. Uma instrução de programação pode ser traduzida em uma ou várias instruções de máquina por um **compilador**.

Exemplos são Java, C, C++, Basic, Fortran

2. Linguagens de Montagem de Baixo Nível

Linguagens de montagem são similares às linguagens de máquina, são mais simples de programar pois possuem poucos comandos e permitem ao programador substituir nomes por números. Linguagens de montagem estão disponíveis em cada família de CPU, e cada instrução de montagem é traduzida em uma instrução de máquina por um programa **montador**.

Nota: Os termos "alto nível" e "baixo nível" são relativos. Originalmente, linguagens de montagem eram consideradas de baixo nível e COBOL, C, etc. eram consideradas de alto nível. Muitos programadores, hoje em dia, podem se referir a estas últimas como linguagens de baixo nível.

5. O Ciclo de Vida de Desenvolvimento de Programas

Programadores não sentam e simplesmente começam a escrever código de uma vez, quando estão tentando fazer um programa de computador. Ao invés disto, eles seguem um planejamento organizado ou metodologia, que quebra o processo em uma série de tarefas.

Este é o ciclo de vida quando se tenta resolver um problema no computador:



Para entendermos o funcionamento deste ciclo na solução de problemas no computador, vamos definir um problema exemplo que iremos resolver passo a passo enquanto discutimos as metodologias para resolução de problemas em detalhe.

5.1. Definir o problema

Geralmente, um programador recebe uma tarefa na forma de um problema. Antes do programa poder ser projetado para resolver um problema em particular, o problema deve, em primeiro lugar, ser bem e claramente definido em termos dos seus requisitos de entrada e saída.

Um problema claramente definido já é metade da solução. Programação de computadores requer que o problema seja primeiro definido antes de se pensar em criar a solução.

Vamos definir o problema exemplo:

“Crie um programa que irá determinar o número de vezes que um nome aparece em uma lista.”

5.2. Analisar o problema

Depois do problema ter sido definido adequadamente, o mais simples e também o mais eficiente e efetivo meio de se resolver será visualizá-lo através de uma representação clara e objetiva.

Geralmente, este passo se dá com a quebra do problema em sub-problemas menores e mais simples.

Problema Exemplo:

Determinar o número de vezes que um nome aparece em uma lista

Entrada para o programa:

Lista de nomes, nome que se deseja procurar

Saída do programa:

O número de vezes que o nome aparece em uma lista

5.3. Projetar e representar o algoritmo

Logo que o problema estiver sido claramente definido, podemos nos concentrar em desenvolver a solução. Na programação de computadores, geralmente é requerido que expressemos a solução passo a passo.

Um Algoritmo é uma especificação clara e não ambígua dos passos necessários para se resolver o problema. Ele pode ser expresso tanto em **linguagem humana** (Inglês, Tagalog e Português), como através de representação gráfica como fluxograma ou através de pseudocódigo, que é um meio termo entre a linguagem humana e a linguagem de programação.

Dado o problema definido na seção anterior, como podemos expressar a solução de uma maneira simples e que possa ser entendida?

Expressando a solução através da linguagem humana:

1. Obter a lista de nomes, vamos chamá-la de **NomeLista**
2. Obter o nome a ser procurado, vamos chamá-lo de **NomeChave**
3. Criar um contador, vamos chamá-lo de **Conta**
4. Pegar cada nome em **NomeLista**
5. Se **NomeChave** for igual ao nome selecionado em **NomeLista**
6. Adicionar 1 a **Conta**
7. Repetir 4 até que todos os nomes já tiverem sido comparados
8. Exibir o valor de **Conta**

Expressando a solução através de um fluxograma:

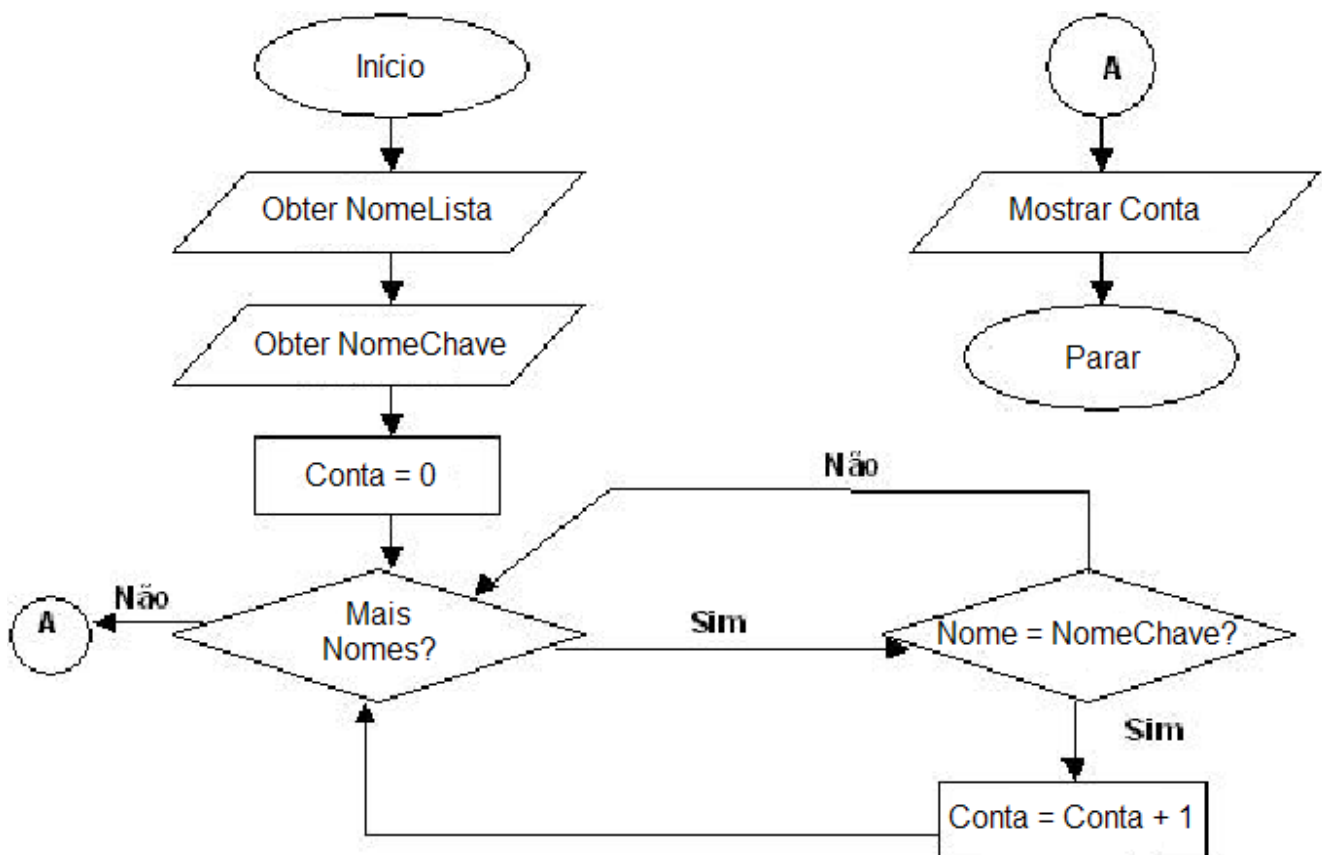


Figura 4: Exemplo de um fluxograma

Expressando a solução através de pseudocódigo:

```

Fazer NomeLista = Lista de Nomes
Fazer NomeChave = o nome a ser procurado
Fazer conta = 0
Para cada nome em NomeLista fazer
  Se nome é igual a NomeChave
    Fazer Conta = Conta + 1
Mostrar Conta



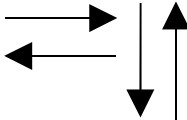
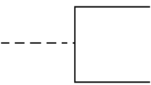
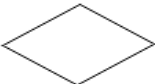


```

Figure 5: Exemplo de pseudocódigo

5.3.1. Símbolos do Fluxograma e o seu significado

Um fluxograma é uma ferramenta de projeto usada para representar graficamente a lógica de uma solução. Os fluxogramas, tipicamente, não mostram comandos de linguagem de programação. Ao invés disto, eles mostram o conceito em Português ou em notação matemática.

Aqui estão algumas dicas dos símbolos mais usados para a criação de fluxogramas. Pode-se utilizar quaisquer símbolos quando criar os seus fluxogramas, desde que use-os de maneira consistente.

Símbolo	Nome	Significado
	Símbolo de Processo	Representa o processo de se executar uma operação definida ou grupo de operações que resultam em mudança de valor, forma ou localização da informação. Também funciona como símbolo padrão quando nenhum outro símbolo estiver disponível.
	Símbolo de Entrada/Saída (E/S)	Representa a função de E/S que faz com que os dados fiquem disponíveis para processamento (entrada) ou para a exibição (saída) das informações processadas.
	Símbolo de Linha	Representa a seqüência de informações disponíveis e operações executáveis. As linhas conectam outros símbolos, e as setas são obrigatórias somente em fluxos com orientação da direita para esquerda e de baixo para cima.
	Símbolo de Anotação	Representa a adição de informação descritiva, comentários, ou notas explicativas para esclarecimentos. A linha vertical e a linha pontilhada podem ser colocadas à esquerda, como mostrado, ou à direita.
	Símbolo de Decisão	Representa a decisão que determina qual das alternativas será seguida.
	Símbolo Terminal	Representa o começo, o final, um ponto de interrupção ou um atraso em um programa.
	Símbolo Conector	Representa qualquer entrada, ou saída, a outra parte do fluxograma. Também serve como um conector fora de página.


<i>Símbolo</i>	<i>Nome</i>	<i>Significado</i>
	Símbolo de Processo Pré-definido	Representa um processo nomeado consistindo de uma ou mais operações ou passos de programa especificados em algum outro lugar.

Tabela 2: Símbolos do Fluxograma

5.4. Codificar e Depurar

Depois de construir o algoritmo, será possível criar o código fonte. Usando o algoritmo como base, o código fonte pode ser escrito usando a linguagem de programação escolhida.

Na maioria das vezes, depois do programador ter escrito o programa, este poderá não estar funcionando 100% no início. O programador deve corrigir o programa no caso de erros (também conhecidos como Erros de Compilação) que ocorrem no programa. Este processo é chamado de **depuração de erros (debug)**.

Existem dois tipos de erros que os programadores poderão encontrar. O primeiro é o erro em tempo de compilação e o outro é o erro em tempo de execução.

Erro em tempo de compilação ocorre se há um erro de sintaxe no código. O compilador irá detectar o erro e o programa nem mesmo compilará. Neste ponto, o programador estará inapto a criar um executável que possa ser executado pelo usuário até que o erro seja corrigido.

Esquecer um ponto-e-vírgula no final de uma instrução ou escrever um comando erroneamente, por exemplo, são erros em tempo de compilação. É algo que o compilador pode detectar como sendo um erro.

Compiladores não são perfeitos e então não podem detectar todos os erros em tempo de compilação. Isso é especialmente verdadeiro para erros de lógica como as repetições (loops) infinitos. Este tipo de erro é chamado de **erro em tempo de execução**.

Por exemplo, a sintaxe do código pode estar correta. Entretanto, ao seguir a lógica do código, o mesmo pedaço de instrução é executado várias e várias vezes, infinitamente. Neste caso, os compiladores não são espertos o suficiente para pegar todos estes tipos de erro em tempo de compilação, conseqüentemente, o programa compila corretamente em um arquivo executável. Entretanto, quando o usuário final roda o programa, o programa (ou mesmo o computador inteiro) congela devido a uma repetição infinita. Outros tipos de erro em tempo de execução são: um valor errado a ser computado, uma instrução errada a ser executada, etc.

6. Sistemas Numéricos e Conversões

Números podem ser representados de várias maneiras. A representação depende do que é chamado de BASE. As que seguem são quatro das representações mais comuns.

6.1. Decimal

Normalmente representamos os números na forma decimal. Números na forma decimal estão na base 10. Isto significa que os únicos dígitos que aparecem são 0-9. Aqui estão alguns exemplos de números escritos na forma decimal:

126_{10} (normalmente escrito somente como 126)
 11_{10} (normalmente escrito somente como 11)

6.2. Binário

Números na forma binária estão na base 2. Isto significa que os únicos dígitos aceitos são 0 e 1. Precisamos escrever a subscrição ₂ para indicar que o número é um número binário. Aqui estão alguns exemplos de números escritos na forma binária:

1111110_2
 1011_2

6.3. Octal

Números na forma octal estão na base 8. Isto significa que os únicos dígitos aceitos são 0-7. Precisamos escrever a subscrição ₈ para indicar que o número é um número octal. Aqui estão alguns exemplos de números escritos na forma octal:

176_8
 13_8

6.4. Hexadecimal

Números na forma hexadecimal estão na base 16. Isto significa que os únicos dígitos aceitos são 0-9 e as letras A-F (ou a-f, minúsculas ou maiúsculas não importam). Precisamos escrever a subscrição ₁₆ para indicar que o número é um número hexadecimal. Aqui estão alguns exemplos de números escritos na forma hexadecimal:

$7E_{16}$
 B_{16}

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal Equivalente	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tabela 3: Números Hexadecimais e sua equivalência para números decimais

Decimal	Binário	Octal	Hexadecimal
126_{10}	1111110_2	176_8	$7E_{16}$
11_{10}	1011_2	13_8	B_{16}

Tabela 4: Sumário dos exemplos

6.5. Conversões


6.5.1. Decimal para Binário / Binário para Decimal

Para converter um número decimal em binário, deve-se dividir continuamente este número por 2 e separar o resto (que será ou 0 ou 1), considerando-o como um dígito da forma binária do número. Obter o quociente e dividir novamente por 2, repetir o processo até que o quociente atinja 0 ou 1. Ao término, agrupar todos os restos começando pelo último resto. O resultado é a forma binária do número.

NOTA: O último dígito menor que o divisor (2) será o primeiro número do resultado.

Por Exemplo:

$$126_{10} = ?_2$$

	Quociente	Resto	
126 / 2 =	63	0	 Escreva nesta direção
63 / 2 =	31	1	
31 / 2 =	15	1	
15 / 2 =	7	1	
7 / 2 =	3	1	
3 / 2 =	1	1	
1 / 2 =		1	

Então, escrevendo os restos de baixo para cima, obtemos o número binário 1111110_2 .

Para converter um número binário para decimal, multiplicar o dígito binário por "2 elevado a posição deste número binário". Adicionar todos os produtos para obter o número decimal resultante.

Por Exemplo:

$$1111110_2 = ?_{10}$$

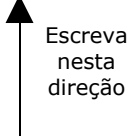
Posição	6	5	4	3	2	1	0	
Dígitos Binários	1	1	1	1	1	1	0	
								$0 \times 2^0 = 0$
								$1 \times 2^1 = 2$
								$1 \times 2^2 = 4$
								$1 \times 2^3 = 8$
								$1 \times 2^4 = 16$
								$1 \times 2^5 = 32$
								$1 \times 2^6 = 64$
								TOTAL: 126

6.5.2. Decimal Para Octal (ou Hexadecimal)/Octal (ou Hexadecimal) para Decimal

Converter números decimais para octal ou hexadecimal é basicamente o mesmo que converter decimal para binário. Entretanto, ao invés de utilizar o 2 como divisor, substituí-lo por 8 (para octal) ou 16 (para hexadecimal).

Por Exemplo (Octal):

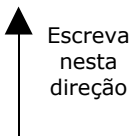
$$126_{10} = ?_8$$

	Quociente	Resto	
126 / 8 =	15	6	
15 / 8 =	1	7	
1 / 8 =		1	

Então, escrevendo os restos de baixo para cima, obtemos o número octal 176₈.

Por Exemplo (Hexadecimal):

$$126_{10} = ?_{16}$$

	Quociente	Resto	
126 / 16 =	7	14 (igual ao dígito hexadecimal E)	
7 / 16 =		7	

Então, escrevendo os restos de baixo para cima, obtemos o número hexadecimal 7E₁₆.

Converter números octais ou hexadecimais é um processo semelhante a converter números de binários para decimal. Para fazer isto, substituímos o número 2 pelo 8 para octal ou pelo 16 para hexadecimal.

Por Exemplo (Octal):

$$176_8 = ?_{10}$$

Posição	2	1	0	
Dígitos Octais	1	7	6	
				6 x 8 ⁰ = 6
				7 x 8 ¹ = 56
				1 x 8 ² = 64
				<hr/> TOTAL: 126

Por Exemplo (Hexadecimal):

$$7E_{16} = ?_{10}$$

Posição	1	0	
Dígitos Hexadecimais	7	E	
			14 x 16 ⁰ = 14
			7 x 16 ¹ = 112
			<hr/> TOTAL: 126

6.5.3. Binário para Octal / Octal para Binário

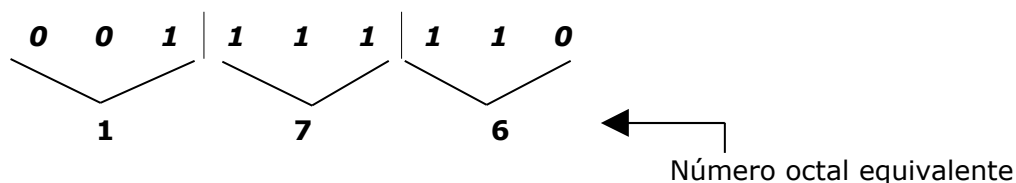
Para converter números binários para octal, partimos o número binário em grupos de 3 dígitos (da direita para esquerda), e o preenchemos com zeros se o número de dígitos não for divisível por 3. Então, convertemos cada partição em seu correspondente dígito octal. A tabela abaixo mostra a representação binária de cada dígito octal.

Dígito Octal	Representação Binária
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Tabela 5: Dígitos Octais e sua representação binária correspondente

Por Exemplo:

$1111110_2 = ?_8$



Converter números octais para binário é o oposto do que foi explicado acima. Simplesmente converta cada dígito octal na sua representação binária (conforme a tabela) e concatene-os. O resultado é a representação binária.

6.5.4. Binário para Hexadecimal / Hexadecimal para Binário

Para converter números binários para hexadecimal, partimos o número binário em grupos de 4 dígitos (da direita para a esquerda), e o preenchemos com zero se o número de dígitos não for divisível por 4. Então convertemos cada partição em seu dígito hexadecimal correspondente. A tabela abaixo mostra a representação binária de cada dígito hexadecimal.

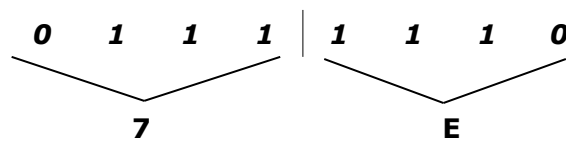
Dígito Hexadecimal	Representação Binária
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011

Dígito Hexadecimal	Representação Binária
C	1100
D	1101
E	1110
F	1111

Tabela 6: Dígitos Hexadecimais e sua representação binária correspondente

Por Exemplo:

$1111110_2 = ?_{16}$



Número hexadecimal equivalente

Converter números hexadecimais para binário é o oposto do que foi explicado acima. Simplesmente converta cada dígito hexadecimal na sua representação binária (conforme a tabela) e concatene-os. O resultado é a representação binária.

7. Exercícios

7.1. *Escrevendo Algoritmos*

Dado o seguinte conjunto de tarefas, crie um algoritmo para realizar cada uma das tarefas abaixo. Escreva os algoritmos usando pseudocódigo ou fluxogramas.

1. Assar pão
2. Acessar o computador
3. Obter a média de três números

7.2. *Conversão de Números*

Converta os números abaixo:

1. 1980_{10} para binário, hexadecimal e octal
2. 1001001101_2 para decimal, hexadecimal e octal
3. 76_8 para binário, hexadecimal e decimal
4. $43F_{16}$ para binário, decimal e octal

Parceiros que tornaram JEDI™ possível



Instituto CTS

Patrocinador do DFJUG.

Sun Microsystems

Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG

Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil

Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec

Suporte e apoio financeiro e logístico a todo o processo.

Borland

Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB

Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.