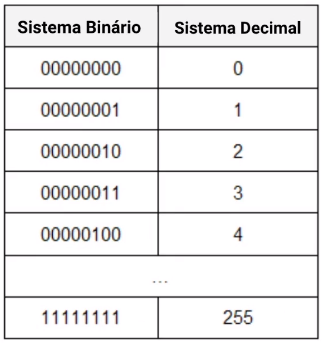
**Arquitetura de Computadores do Século XXI**

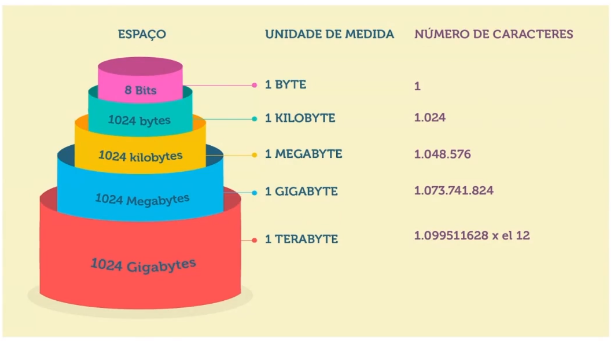
**Sistemas de Numeração**

Bit = “0, aberto” ou “1, fechado”.

Byte = 8 vezes o bit. Ex: A= 10010010, a= 11001100



**Grandezas de Armazenamento de Informação**



**Sistemas de Numeração**

A quantidade de algarismos disponíveis num sistema de numeração designa-se de **base**.

Alguns sistemas de numeração.

- Decimal (base 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

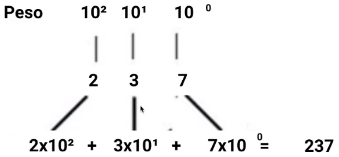
- Binário (base 2): 0, 1.

- Octal (base 8): 0, 1, 2, 3, 4, 5, 6, 7.

- Hexadecial (base 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

**Sistema Decimal**

No sistema decimal cada algarismo tem um valor posicional, ou seja, cada algarismo tem um peso de acordo com a sua posição na representação do valor.



**Sistema Binário**

É o sistema mais utilizado por máquinas, uma vez que os sistemas digitais trabalham internamente com dois estados (ligado/desligado, verdadeiro/falso, aberto/fechado). O sistema binário utiliza os símbolos: 0, 1, sendo cada símbolo designado por bit (**binary digit**).

**Sistema Octal**

É um sistema de numeração de base 8, ou seja, recorre a 8 símbolos (0, 1, 2, 3, 4, 5, 6, 7) para representação de um determinado valor.

**Sistema Hexadecimal**

Muito utilizado na programação de microprocessadores, especialmente nos equipamentos de estudo e sistemas de desenvolvimento. Utiliza os símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 do sistema decimal e as letras A, B, C, D, E, F.

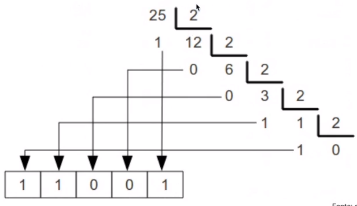
**Equivalências**: A= 10, B= 11, C= 12, D= 13, E= 14 e F= 15.

**Conversões de Base Decimal**

**Decimal > Binário**

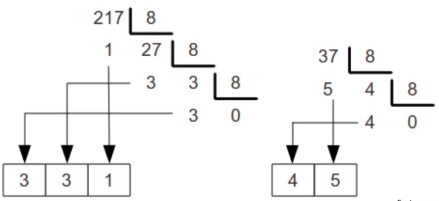
Dividir sucessivamente por 2 o número decimal e os quocientes que vão sendo obtidos, até que o quociente de uma das divisões seja 0.

O resultado é a sequência de baixo para cima de todos os restos obtidos.



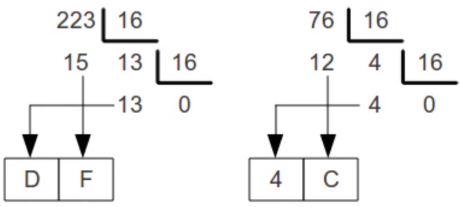
**Decimal > Octal**

Divisões sucessivas por 8. O resultado é a sequência de baixo para cima de todos os restos obtidos (a mesma coisa que binário).



**Decimal > Hexadecimal**

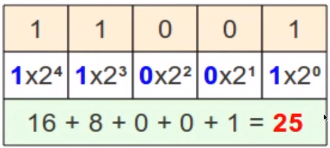
Divisões sucessivas por 16.



**Conversões de base binária, octal e hexadecimal**

**Binário > Decimal**

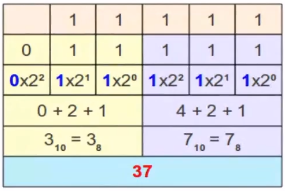
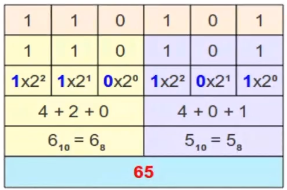
Basta calcular a soma de cada um dos dígitos do número binário multiplicado por 2 (que é a sua base) elevado à posição colunar do número, que, da direita para a esquerda começa com 0.



**Binário > Octal**

Separa-se os dígitos do número binário em grupos de 3 bits da direita para a esquerda. Em seguida transforma-se cada grupo individual de 3 bits em octal. Ao final, une-se os resultados:

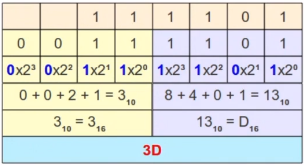
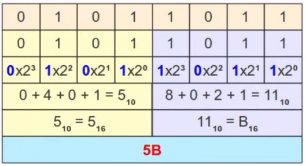
Caso o número de dígitos do número binário não seja múltiplo de 3, completa-se os dígitos à esquerda com zeros (0):



**Binário > Hexadecimal**

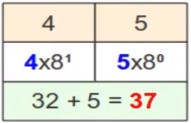
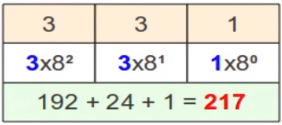
Separa-se o número binário em grupos de 4 bits, da direita para a esquerda. Em seguida, transforma-se cada grupo de 4 bits em hexadecimal. Ao final, simplesmente une-se os resultados em um só:

Caso o número de dígitos do número binário não seja múltiplo de 4, completa-se os dígitos à esquerda com zeros(0):



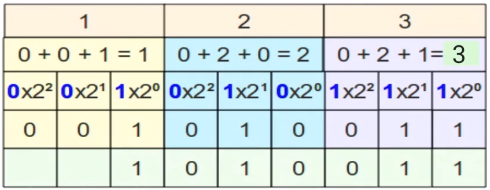
**Octal > Decimal**

Obtida através da soma dos dígitos do número octal multiplicados pela base 8 elevada à posição colunar do dígito, começando em 0 da direita para a esquerda:



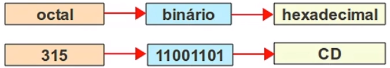
**Octal > Binário**

Decompõem-se o número octal diretamente em binários de 3 digitos. Os zeros mais à esquerda do resultado binário podem ser omitidos:



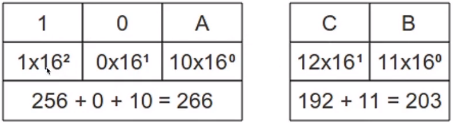
**Octal > Hexadecimal**

Transforma-se primeiro o octal em binário e em seguida o binário em hexadecimal:



**Hexadecimal > Decimal**

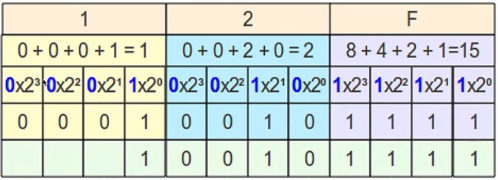
Realizada através da soma dos dígitos hexadecimais multiplicados pela base 16 elevada à posição colunar contando da direita para a esquerda, começando em 0, de forma semelhante à conversão de binários em decimais:



Note que os caracteres que definem os dígitos hexadecimais A, B e C foram substituídos pelos valores equivalentes em decimais 10, 11 e 12 de acordo com a tabela da lição anterios para a realização do cálculo.

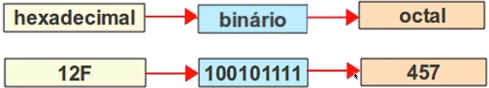
**Hexadecimal > Binário**

Decompõem-se o número hexadecimal diretamente em binários de 4 dígitos. Os zeros mais à esquerda do resultado binário podem ser omitidos:



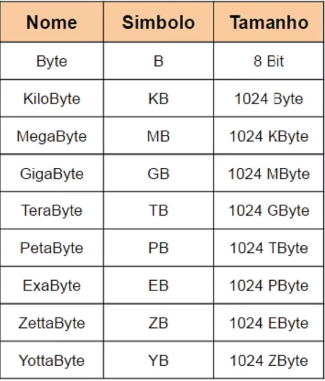
**Hexadecimal > Octal**

Transforma-se primeiro o hexadecimal em binário e em seguida o binário em octal:

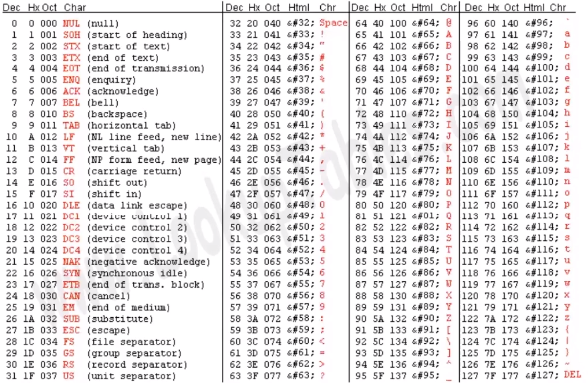


**Resumos**

**Grandezas**



**Tabela ASCII**



**# Conversor de bases:** <https://www.freetool.dev/pt/conversor-base-numerica>

**Algoritmos**

**Algoritmo**

É uma série ordenada de passos não ambíguos, executáveis. Um algoritmo é uma sequência de instruções ou comandos realizados de maneira sistemática com o objetido de resolver um problema ou executar uma tarefa.

Algoritmos são como uma receita de bolo: uma sequência de ações que devem ser executadas até que o objetivo final – o bolo pronto – seja atingido.

**Algoritmo clássico: Cubo Mágico**

Cubo de Rubik, também conhecido como Cubo Mágico, é um quebra-cabeça tridimensional, inventado pelo professor de arquitetura húngaro Ernõ Rubik em 1974

**Algoritmo clássico: Cubo Mágico**

Considere três hastes que servem de suporte a discos de diferentes tamanhos, os menores sempre sobre os maiores. Pode-se mover cada disco para qualquer haste desde que um disco maior nunca fique sobre um disco menor. O objetivo é transferir os três discos para uma outra base.

**Diferentes Estruturas**

Em um algoritmo ou mesmo programa de computador, são várias as estruturas que utilizam para controlar o fluxo de execução dos comandos e funções. Essas estruturas determinam o que será executado, como e quando será executado

**Eficiência de Algoritmos**

**Algoritmos Eficientes**

A análise de algoritmo fornece uma medida objetiva de desempenho proporcional ao tempo de execução do algoritmo.

1° facilidade de compreensão, codificação e depuração;

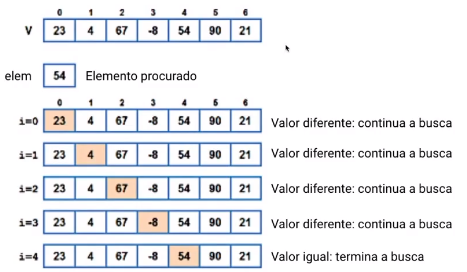
2° eficiência na utilização dos recursos do computador e rapidez, através de seu tempo de execução;

**Buscas e Ordenações**

Se você pensar bem, quase tudo que fazemos em termos de computação é busca de elementos em uma lista ou ordenação de elementos em uma lista! Métodos de **Busca** e **Ordenação** são a essência de muitos problemas e soluções em Computação.

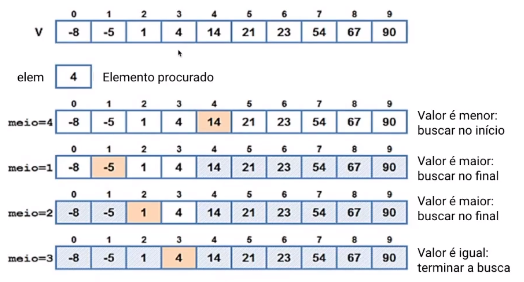
**Busca Sequencial**

A busca sequencial é um algoritmo pouco eficiente para encontrar um item em uma lista grande de elementos. No pior caso, todos os elementos devem ser visitados!



**Busca Binária**

A busca binária é um eficiente algoritmo para encontrar um item em uma lista ordenada de itens. Ela funciona dividindo repetidamente pela metade a porção da lista que deve conter o item, até reduzir as localizações possíveis a apenas uma.

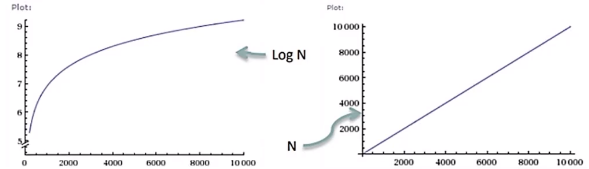


**Busca Sequencial x Busca Binária**

Se a lista não está ordenada, a busca binária não se aplica.

A busca sequencial é muito ineficiente para conjuntos grandes de valores.

Costuma-se dizer que a busca sequencial tem complexidade n e que a busca binária tem complexidade log n, fazendo muito menos comparações, torna-se mais eficiente.



**Linguagens de Programação**

**BASIC**

É uma linguagem historicamente importante que ajudou a popularizar a prática de programação. Alguns dos primeiros computadores pessoais vinham com a linguagem BASIC instalada no hardware convidando novos usuários a começar a programar. Várias derivações de BASIC estão ou estiveram disponíveis, entre elas SmallBasic, VisualBasic, entre outras.

**COBOL**

Muitos sistemas na área bancária foram escritos em Cobol e permanecem em uso até hoje. A formação em COBOL não é comum e por isso mesmo os programadores em COBOL costumam ser muito valorizados.

**PYTHON**

Machine Learning e Extração de Dados são implementados com Python.

**ASSEMBLY**

É a linguagem que leva à alta performance da máquina ainda em padrão legível pelas pessoas. É usada em partes de um programa muito sensíveis à performance. Encontrada em sistemas operacionais e “engine” de jogos, por exemplo.

**C**

Talvez a linguagem de programação mais importante no mundo. Sistemas operacionais tais como Windows, MacOs, iOs, e Android são escritos nela, bem como navegadores e “engine” de jogos. Influenciou dezenas de outras linguagens. Próxima da linguagem Assembly Language permite alta performance do software.

**JAVA**

Foi desenvolvido pela SUN Microsystems no início de 1990. É “orientada a objetos”, escalável e portável. Tem muitas semelhanças com C e C++. É a base de diversos sistemas como Android, possuindo uma comunidade forte e ativa na internet. Muitas empresas tem sistemas desenvolvidos em Java, fazendo com que a procura por profissionais que saibam a linguagem ainda seja expressiva.

**RUBY**

Sintaxe simples e fácil. Inspirada em linguagens como Perl, Smalltalk, Eiffel, Ada e Lis. Muito popular entre as startups, é famosa por ser usada em aplicações mundialmente reconhecidas, como Airbnb, Twitter e Github.

**Javascript**

Pode ser usada tanto no front quanto no back-end, sendo uma das linguagens mais versáteis que existem. É a linguagem majoritária para desenvolvimento Web e dificilmente um programador não terá contato com ela alguma vez na vida no mercado de trabalho.

**PHP**

É a linguagem mais famosa para criar backends de websites. Facebook e WordPress foram, em parte, escritas nela.

**SWIFT**

Desenvolvimento de softwares para a Apple.

**C#**

É portátil e simples. Desenvolvida e mantida pela Microsoft, é uma linguagem muito conhecida e fortíssima no cenário de desenvolvimento de jogos. Baseada nas linguagens C++, Java e Object Pascal.

**C++**

Semelhante à linguagem C.

Essa linguagem foi um esforço de se fazer uma linguagem fácil para a construção de grande projetos e ainda rápida e eficiente.

**Classificação de Linguagens**

**Alto Nível e Baixo Nível**



**Paradigma Imperativo Procedural**

Primeiro paradigma de programação baseado no modelo clássico de Von Neumann;

-Atribuições, sequências de comandos, laços de repetição e comandos condicionais fazem parte deste paradigma.

-Abstração procedural é sua principal característica.

Ex: C, Cobol, Fortran, Pascal, Ada, etc.

**Paradigma Imperativo Orientado a Objetos**

Um programa é constituído de vários objetos que trocam mensagens entre si.

-Objetos de dados e não passivos como no paradigma imperativo;

-Definição de classes de objetos, herança e passagem de mensagens caracterizam este paradigma;

Ex: C++, C#, Java, SmallTalk, etc.

**Paradigma Declarativo Funcional**

O problema é modelado por um conjunto de funções matemáticas, cada uma com um espaço de entrada e um resultado, usada tradicionalmente em IA.

-As funções interagem entre si, utilizando a composição funcional;

Ex: LISP, Haskell, ML, etc.

**Paradigma Declarativo Lógico**

Declarativo – descreve o problema e o que se deseja fazer e não como fazer (programação imperativa e OO).

-Conhecida como baseada em regras;

Ex: Prolog.

**Quanto à Estrutura de Tipos**

-Fracamente tipada: Tipo das variáveis não é definido, podendo modificar-se. Ex: PHP.

-Fortemente tipada: Tipo das variáveis é predefinido e imutável. Ex: Java, Ruby.

-Dinamicamente tipada: Tipo das variáveis é definido em tempo de execução. Ex: Python, Ruby.

-Estaticamente tipada: Tipo das variáveis é definido em tempo de compilação. Ex: Java, C.

**Quanto ao Grau de Abstração**

O grau de abstração funciona como uma escala para linguagens: quanto mais abaixo mais próximo da linguagem de máquina, e quanto mais alto, mais próxima da linguagem dos seres humanos.

-Baixo nível: Possui símbolos que representam o código de máquina propriamente. Ex: Assembly.

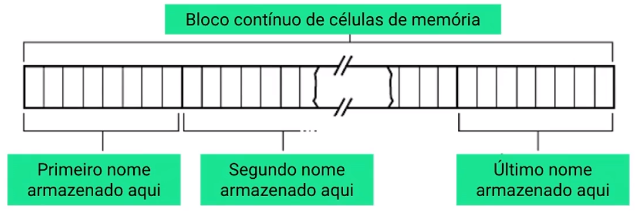
-Médio nível: Possui símbolos que podem ser diretamente traduzíveis para código de máquina, mas também possui símbolos que precisam ser processados por um compilador. Ex: C#.

-Alto nível: Possui símbolos complexos que precisam de interpretação de um compilador antes de serem transformados em linguagem de máquina. Ex: Java, Javascript, Python, Ruby.

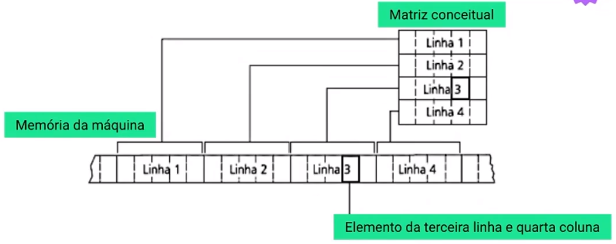
**Abstração de Dados**

A forma como o usuário ou programador lida com os dados não corresponde à forma como os dados estão realmente armazenados na memória do computador ou como eles são realmente.

Chamamos de abstração de dados, estruturas muito úteis que “mascaram” os dados para visualizar e manipular dados de forma a tornar as soluções de uso e programação mais intuitivas.

**Vetor (array) ou lista contígua**

É uma lista linear na qual a ordem lógica dos elementos (a ordem “vista” pelo usuário) é a mesma ordem física (em memória principal) dos elementos. Isto é, elementos vizinhos na lista estarão em posições vizinhas de memória.

**Matriz (4l x 5c) Armazenada Segundo a Sequência de Linhas**

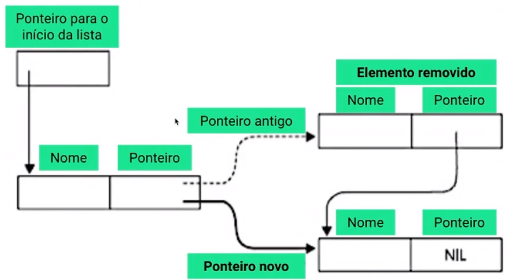
Matriz é uma tabela organizada em linhas e colunas no formato m x n, onde m representa o número de linhas (horizontal) e n o número de colunas (vertical). A função das matrizes é relacionar dados numéricos.

**Listas Ligadas**

É uma estrutura que corresponde a uma sequência lógica de entradas ou nós.

Em uma lista ligada há um ou dois pontos conhecidos de acesso – normalmente o topo da lista (seu primeiro elemento) e, eventualmente, o fim da lista (seu último elemento). Cada nó armazena também a localização do próximo elemento na sequência, ou seja, de seu nó sucessor. Desse modo, armazenamento de uma lista não requer uma área de contígua de memória.

Nesta estrutura os nomes não ocupam uma área contígua de memória, por isso cada elemento da estrutura tem o dado (nome) e um apontador para o próximo nome.

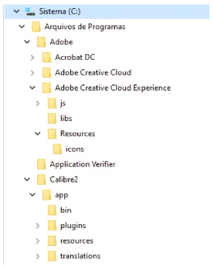


**Lista Ligada: Eliminando um Elemento**

Observe que a remoção de um elemento implica rearranjo de ponteiro e só.

**Lista Ligada: Inserindo um Elemento**

O mesmo acontece na inserção, só mudança de ponteiros.

**Árvores e Exemplos**

**Aplicação**

Na computação, o uso das estruturas em árvore é comum:

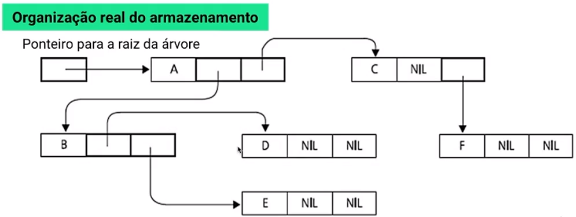
**Árvores**

Em uma árvore, a organização dos elementos se dá de forma hierárquica, existindo um elemento que fica no topo da árvore, chamado de raiz e os elementos subordinados a ele, os nós filhos.

Cada nós filho pode conter zero, um ou mais de um nó filho.

Estruturas em árvore possibilitam algoritmos mais eficientes!

**Árvore Implementada por um Sistema de Listas Ligadas**



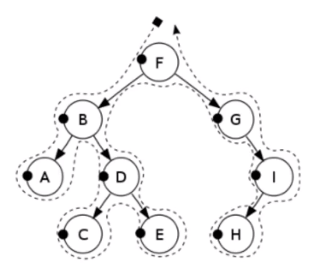
**Árvore de Pesquisa Binaria**

É uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo.

Um exemplo: Encontrar todas as repetições numa lista de números. Uma forma para fazer isto é comparar cada número com todos que o precedem.

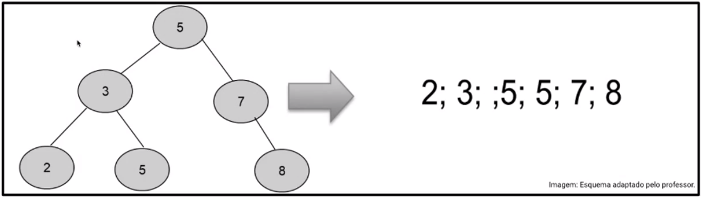
**Percursos em Árvore Binária**

Veja na imagem como algoritmos PreOrdem, InOrdem, PosOrdem geram saídas diferentes!



**Exemplo: Percorrendo uma Árvore em Ordem**

A chave da raiz de uma sub-árvore é impressa entre os valores de sua sub-árvore esquerda e sub-árvore direita.



**Pilha**

São estruturas de dados do tipo LIFO (last-in-first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado.

Assim, uma pilha permite acesso a apenas um item de dados – o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

São muito importantes quando precisamos armazenar um conjunto de dados para ser utilizado em um determinado software.

**Fila**

São estruturas de dados do tipo FIFO (first-in first-out), onde o primeiro elemento a ser inserido, será o primeiro a ser retirado, ou seja, adiciona-se itens no fim e remove-se do início.

**Engenharia de Software**

Uma disciplina que reúne metodologias, métodos e ferramentas a serem utilizadas desde a percepção de um problema até o momento em que o sistema desenvolvido deixa de ser operacional, visando resolver problemas inerentes a processo de desenvolvimento e ao produto de software.

Visando: - baixar custo.

- produção mais rápida.

- melhoria da qualidade.

**Requisitos**

Correspondem ao conjunto de necessidades do usuário em relação ao sistema. Requisitos podem ser funcionais (o que o sistema deveria fazer) e não funcionais (restrições, isto é, o que o sistema não deveria fazer).

**Extração de Requisitos**

É uma das tarefas mais desafiadoras na área de desenvolvimento de sistemas, envolve várias técnicas tais como entrevistas, formulários, workshops, documentação, análise de cenários, etc. A extração de requisitos pode resultar em um documento de requisitos definido pelo padrão IEEE 830.

**Padrão IEEE830**

#Requisitos Relacionados com Interfaces Externas

Interfaces com o Usuário

Interfaces com Hardware

Interfaces com Outros Sistemas de Software

Interfaces de Comunicação

#Requisitos Funcionais

Requisito Funcional 1\*

Requisito Funcional 2\*

...

#Requisitos de Desemprenho

#Requisitos de Projeto

#Outros Requisitos

**Técnicas de Levantamento de Requisitos: Entrevistas**

É a mais utilizada por ser simples e de baixo custo;

É recomendado que o entrevistador prepare as perguntas antecipadamente;

Pode ter uma longa duração;

É difícil que o entrevistador fique “neutro”, ou seja, não utilize sua experiência para antecipar uma solução e induzir o usuário e aceita-la.

**Protótipos**

É uma aplicação ou sistema em fase de teste.

**Ciclos de Vida**

É uma estrutura que indica processos e atividades envolvidas no desenvolvimento, operação e manutenção de um software, abrangendo de fato toda a vida do sistema. Neste ciclo, existem modelos que definem como o software será desenvolvido, lançado, aprimorado e finalizado.

Ciclo de vida é o conjunto de etapas para o desenvolvimento da aplicação desde o levantamento das necessidades do usuário até a entrega do produto e mesmo após a entrega o ciclo pode continuar enquanto o desenvolver estiver trabalhando no Sistema.

Análise > Projeto > Codificação > Testes > Implementação

**Modelos de Ciclo de Vida**

São descrições abstratas do processo de desenvolvimento, mostrando as principais atividades e informações usadas na produção e manutenção de software, bem como a ordem que as atividades são executadas.

Análise de Requisitos > Design > Código > Testes > Manutenção

**Modelo Espiral**

É um processo evolucionário, ou seja, adequado para softwares que precisam passar por inúmeras evoluções na medida que o desenvolvimento acontece. Diferente do Modelo Incremental, que entrega partes prontas uma de cada vez, o Modelo Espiral faz sucessivos refinamentos. Outras novidades são os conceitos de Prototipagem e Gerenciamento de Riscos. O Modelo em Espiral “combina prevenção e tolerância a mudanças, assume que mudanças são um resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua redução”. O modelo é “uma abordagem realista do desenvolvimento de sistemas e softwares de grande porte usando a prototipagem como mecanismo de redução de riscos”.

1° São identificados objetivos específicos.

2° Riscos são detectados e atividades são acionadas para reduzir riscos.

3° Um modelo de desenvolvimento é escolhido.

4° O projeto é revisado e a próxima fase da espiral é planejada.

**Garantia de Qualidade**

Significa a definição de procedimentos, processos e padrões que visam reforçar que a qualidade de software seja atingida. A garantida de qualidade também inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicadas após o produto ter sido entregue por uma equipe de desenvolvimento.

**Verificação e Validação**

Verificação: Consiste em avaliar se existem falhas e problemas com o software (seja no código, nas funcionalidades, interface, dentre muitos outros detalhes) antes que ele seja entregue ao cliente ou disponibilizado para o público final.

Validação: A validação de software é um processo que comprova documentalmente que o sistema cumpre com as funções das quais foi designado, em conformidade com as especificações dos requisitos do usuário e com a garantia de segurança e rastreabilidade de informações.

Podemos sintetizar que as quatro atividades básicas do desenvolvimento do design de interação incluem:

1. Identificar necessidades e requisitos do usuário.
2. Desenvolver designs alternativos.
3. Construir versões interativas dos designs.
4. Avaliar designs.

**Gerência de Projetos: Gráficos de Atividades**

Significa simplesmente a definição de procedimentos, processos e padrões que visam reforçar que a qualidade de software seja atingida. Em outros casos, a garantia de qualidade também, inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicadas após o produto ter sido entregue por uma equipe de desenvolvimento.

**Testes do Tipo Caixa Branca**

Nesse processo, o usuário analisa por qual caminho ocorre o fluxo de dados e é possível verificar se há a passagem correta em todas as condições esperadas.

**Testes do Tipo Caixa Preta**

O usuário não tem acesso ao código fonte e nem a sua estrutura. Como é baseado nos requisitos funcionais, ele também é chamado de teste funcional.

**Sistemas Operacionais**

**Modelo de Von Neumann**

Os componentes do modelo de von Neumann são:

- Unidade de controle (que busca cada instrução dos programas na memória e executa sobre os dados de entrada);

- Unidade lógica e aritmética ULA (parte da unidade central de processamento);

- Memória (para armazenar dados e programas);

- Os dispositivos de entrada e saída interagem com o modelo, mas não fazem parte dos componentes principais;

**CPU – Unidade Central de Processamento**

O cérebro do computador!

**Unidade Lógica e Aritmética (ULA)**

A unidade lógica e aritmética realiza ações indicadas nas instruções, executando operações numéricas (aritméticas) e não numéricas (lógicas), além da preparação de informações para desvios do programa.

A Unidade de Controle (UC) e a unidade lógica e aritmética (ULA) formam a unidade central de processamento (UCP), ou simplesmente processador.

**Unidade Lógica e Aritmética (ULA)**

Realiza operações aritméticas e operações lógicas sobre um ou mais operandos. Exemplos de operações realizadas pela ULA: soma de dois operandos; negação de um operando; inversão de um operando; AND (“E” lógico) de dois operandos; OR (“OU” lógico) de dois operandos; deslocamento de um operando para a esquerda ou para a direita; rotação de um operando para a esquerda ou para a direita.

As operações da ULA são, geralmente, muito simples. Funções mais complexas, exigidas pelas instruções da máquina, são realizadas pela ativação sequencial das várias operações básicas disponíveis.

**Memória**

A memória de um sistema de computador tem a função de armazenar dados e instruções; é organizada em posições, que podem ser visualizadas como elementos em uma matriz.

Cada elemento da memória tem um endereço. Assim, pode-se falar de uma memória que tenha x posições: cada posição pode ser referenciada diretamente de acordo com a sua colocação na sequência.

Por exemplo, se uma memória tem 4096 posições, existem posições de memória 0, 1, 2, 3, ..., 4094 e 4095.

Um registrador é como uma pequena unidade de memória.

Quando um endereço de memória aparece nos circuitos de controle conectados à memória, o conteúdo (o valor que está na posição) será trazido da memória para os circuitos da unidade de processamento ou a informação na unidade de processamento será armazenada na memória, dependendo do trabalho associado com aquele endereço.

**Instruções**

Em um computador são executadas em uma sequência determinada por suas posições de memória.

Na maioria dos computadores, instruções e dados são distribuídos em posições de memória.

**Drivers (Controladores)**

Os softwares que permitem a combinação do hardware, que é a parte física do computador, placa mãe e com outros dispositivos com o sistema operacional.

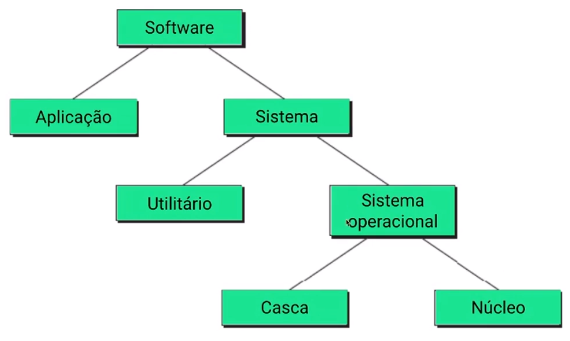
É o driver que converte as informações do hardware para o sistema operacional do computador.

É ele que cuida quando ao se abrir um requerimento, o processo seja executado, sendo permitida a interação do software com o dispositivo.

**Programa**

É constituído de uma sequência predeterminada de instruções, que deve ser seguida para que seja alcançado o objetivo computacional. O programa e os dados correspondentes estão armazenados na memória da máquina; o conjunto de instruções (ou programa) deve ser interpretado para a realização do processamento, isto é, a informação codificada correspondente às ações e aos operandos precisa ser entendida e então processada.

**Uma Hierarquia do Software**



**Sistema Operacional**

Visa abstrair o acesso e gerenciar os recursos de hardware, provendo aos aplicativos um ambiente de execução no qual o acesso aos recursos se faz através de interfaces simples, independentes das características e detalhes de baixo nível, e no qual os conflitos no uso do hardware são minimizados. Os sistemas operacionais podem ser classificados segundo diversos parâmetros e perspectivas, como tamanho, velocidade, suporte a recursos específicos, acesso à rede, etc.

**Para que serve um sistema operacional**

Cada computador possui normalmente um só processador.

O uso do processador deve ser distribuído entre os programas presentes no sistema, de forma que cada um deles possa executar na velocidade adequada para cumprir suas funções sem prejudicar os outros.

Também a memória RAM deve ser gerenciada e distribuída de forma justa entre as aplicações.

Gerenciar as impressões em computadores em rede, por exemplo, é uma tarefa necessária que o sistema operacional faz.

**Tipos de sistemas operacionais**

Um OS visa abstrair o acesso e gerenciar os recursos de hardware, provendo aos aplicativos um ambiente de execução abstrato, no qual o acesso aos recursos se faz através de interfaces simples, independentes das características e detalhes de baixo nível, e no qual os conflitos no uso do hardware são minimizados.

Os OS’s podem ser classificados segundo diversos parâmetros e perspectivas, como tamanho, velocidade, suporte a recursos específicos, acesso à rede, etc.

**Tipos de sistemas operacionais**

**Desktop**:  
Um sistema operacional ‘de mesa’ é voltado ao atendimento do usuário doméstico e corporativo para a realização de atividades corriqueiras. Suas principais características são a interface gráfica, o suporte à interatividade e a operação em rede.

Exemplo: Sério que você queria ver um exemplo disso?

**Servidor**:

Deve permitir a gestão eficiente de grandes quantidades de recursos (disco, memória, processadores), impondo prioridades e limites sobre o uso dos recursos pelos usuários e seus aplicativos. Normalmente um sistema operacional servidor também tem suporte a rede e multiusuários.

**Distribuído**:

Os recursos de cada máquina estão disponíveis globalmente, de forma transparente aos usuários. Ao lançar uma aplicação, o usuário interage com sua janela, mas não sabe onde ela está executando ou armazenando seus arquivos: o sistema é quem decide, de forma transparente.

**Multiusuário**:

Deve suportar a identificação do proprietário de cada recurso dentro do sistema (arquivos, processos, áreas de memória, conexões de rede) e impor regras de controla de acesso para impedir o usu desses recursos por usuários não autorizados. Essa funcionalidade é fundamental para a segurança dos sistemas operacionais de rede e distribuídos. Grande parte dos sistemas atuais são multiusuários.

**De Rede**:

Um sistema operacional de rede deve possuir suporte à operação em rede, ou seja, a capacidade de oferecer às aplicações locais recursos que estejam localizados em outros computadores da rede, como arquivos e impressoras. Ele também deve disponibilizar seus recursos locais aos demais computadores, de forma controlada. A maioria dos sistemas operacionais atuais oferece esse tipo de funcionalidade.

**Embutido**:

É dito embutido (*embedded*) quando é construído para operar sobre um hardware com poucos recursos de processamento, armazenamento e energia. Aplicações típicas desse tipo de sistema aparecem em máquinas industriais e automotivas, equipamentos eletrônicos de uso doméstico (leitores de DVD, eletrônicos, etc.).

**Ciclo de Execução**

Quando a CPU quer ler uma palavra, ou da memória ou de uma porta de E/S, ela coloca o endereço necessário nas linhas de endereço do barramento e, então, envia um sinal READ em uma linha de controle do barramento. Uma segunda linha de sinal é usada para dizer se é necessário espaço de E/S ou espaço de memória. Se for espaço de memória, a memória responderá a requisição. Se for espaço de E/S, é o dispositivo de E/S que responderá.

**Processos na CPU**

**Novo**:

Um novo processo foi adicionado à fila de execuções, então a CPU deverá buscar os dados necessários para executar o processo.

**Pronto**:

O processo encontra-se pronto para ser executado, nesse ponto ele pode ser enviado para a CPU pois os dados básicos para a execução já foram selecionados.

**Em Execução**:

O processo está na CPU sendo executado.

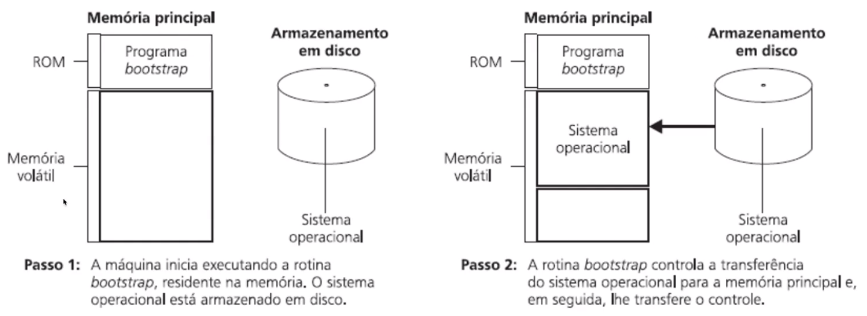
**Em Espera**:

O processo executou alguma operação de Entrada e Saída, ou depende de algum dado que deve ser fornecido por outro processo ou que não foi previamente carregado.

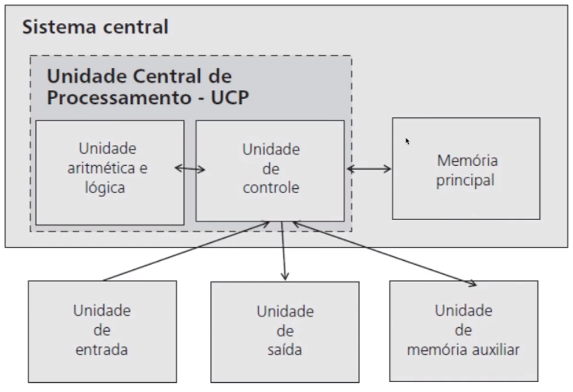
**Terminado**:

O processo encerrou seu ciclo de execução.

**Rotina Bootstrap**



**Esquema Simplificado de um Computador**



**Redes de Computadores**

**O que são as Redes de Computadores**

Conjunto de dispositivos autônomos e interconectados com a finalidade de trocar dados por meio de uma única tecnologia.

**O Início**

Sistema sAgE (Semi-Automatic Ground Enviroement) foi desenvolvido pela IBM para detectar bombardeios russos durante a Guerra Fria. Pioneiro na exploração do emprego de computadores em rede. Ao todo, o sistema contava com vinte e três centros de computação distribuídos na América do Norte, os quais se comunicavam em tempo real com estações de radares e aeronaves de contra-ataque para reagir contra eventuais situações de ameaça. O sistema operou entre as décadas de 1950 e 1980.

Nas primeiras décadas de existência, os sistemas computacionais estavam altamente centralizados.

Presença de “grandes” computadores.

**Contribuíram para o surgimento das redes:**

Instalações de grandes redes telefônicas;

Invenção do rádio e da televisão;

Surgimento e crescimento da indústria de computadores e;

Surgimento e lançamento de satélites;

**Meios de Transmissão**

Corresponde ao meio físico que será utilizado para realizar a comunicação entre hosts e aplicações.

- Cabos; Placas; Modems; Roteadores; Etc.

Redes de acesso: Rede física que conecta um sistema final ao primeiro roteador de um caminho partindo de um sistema final até outro qualquer. Enlaces de comunicação.

Ethernet: Velocidades de cerca de 100Mbits/s 1Gbits/s e 100Gbits/s (na rede de acesso).

**Classificação por Área Geográfica**

Redes Pessoais (Personal Area Network): Redes de curtíssimo alcance, normalmente alguns poucos metros.

Redes Local ou LAN (Local Área Networks): Em que a abrangência de uma rede não ultrapassa algumas dezenas de metros, situando-se, normalmente, dentro de um edifício.

Redes de Longa Distância ou WAN (Wide Área Networks): Consiste normalmente em diversas redes locais entre si, abrangendo um conjunto de edifícios vizinhos, como por exemplo vários países ou até a totalidade do globo (como é o caso da Internet).

Redes Metropolitanas ou MAN (Metropolitan Area Networks): Redes que abarcam a área de uma grande cidade ou região urbana, interligando determinadas entidades ou instituições que necessitam de manter entre si um sistema de comunicações de dados, como por exemplo as entidades administrativas ou policiais de uma grande.

**Protocolos**

Definem os formatos, a ordem das mensagens enviadas e recebidas pelas entidades de rede e as ações a serem tomadas na transmissão e recepção de mensagens.

HTTP (HyperText Transfer Protocol): É o protocolo utilizado para controlar a comunicação entre o servidor de Internet e o browser, ou seja, serve de suporte à World Wide Web. É o que nos permite escrever na Barra de Endereços do nosso browser um endereço URL (Uniform Resource Locator) e rapidamente receber a página Web correspondente.

FTP (File Transfer Protocol): É o protocolo utilizado para a transferência de arquivos. Serve para fazermos o download ou upload de arquivos de ou para servidores FTP, aquele cujo endereço começa com [ftp://](NULL).

SMTP (Simple Mail Transfer Protocol): É o protocolo usado para enviar mensagens de correio eletrônico dos computadores clientes para os servidores e também para transferir mensagens entre servidores. Relacionados com este, existem também o IMAP (Internet Message Access Protocol), permite a procura de mensagens por palavra-chave no servidor e a seleção das mensagens a retirar e o POP3 (Post Office Protocol versão 3), para retirar mensagens do servidor para o nosso computador, o cliente.

Telnet (Emulação de Terminal): É o protocolo associado ao serviço de acesso remoto a outros computadores.

TCP (Transmission Control Protocol): É o protocolo utilizado no transporte de informação entre os computadores emissor e receptor e garante a segurança dessa mesma informação.

UDP (User Datagram Protocol): É um protocolo de transporte de informação entre os computadores emissor e receptor, no entanto, não garante a segurança da informação transmitida. Este protocolo não é tão fiável como o TCP pois apenas estabelece a ligação e envia a informação, assim este é considerado mais rápido, mas menos eficiente.

DNS (File Transfer Protocol): É o protocolo utilizado para associar a cada endereço IP um nome, pois desta forma é mais fácil de ser memorizado pelos usuários. Por exemplo, [www.portoeditora.pt](http://www.portoeditora.pt) corresponde ao endereço IP 192.16831.32.

IP (Internet Protocol): É o protocolo responsável por estabelecer a ligação entre os computadores emissor e receptor para que a informação não se perca na rede.

ICMP (Internet Control Message Protocol): É o protocolo responsável pelo envio de mensagens para responder a pacotes de dados que não foram entregues corretamente. Desta forma é enviada uma mensagem ICMP e volta a ser enviado o pacote de dados não recebido.

**Arquitetura de Redes**

Cliente/Servidor: A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem, os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes.

Ponto a Ponto (Peer-To-Peer/P2P): Arquitetura de redes de computadores onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central.