# Introdução ao Cálculo Numérico e Computacional

## Introdução

O Cálculo Numérico consiste na obtenção de soluções aproximadas de problemas de Álgebra Linear e Não-Linear, Estatística e Análise de Dados, Cálculo Diferencial e Integral e outros métodos matemáticos, utilizando métodos numéricos. Com a popularização de computadores de baixo custo e de alta capacidade de processamento, praticamente todas as atividades de Engenharia tem feito uso cada vez mais intensivo dos métodos e técnicas computacionais na resolução de problemas reais, para os quais as soluções manuais são impraticáveis e/ou imprecisas.

Desta forma, o uso do computador como ferramenta de trabalho de cálculo numérico requer o entendimento dos seus princípios de operação e de como eles interferem nos resultados obtidos. Geralmente, é aceito como verdade que computadores não erram e que são os usuários é que cometem enganos que levam ao mal funcionamento do computador. Na realidade, o computador, como dispositivo de cálculo numérico, "comete" erros devido às suas características intrínsecas e o papel do usuário é quantificar esses erros e encontrar formas de, se não eliminá-los, pelo menos minimizá-los.

## *Hardware* e *Software*

Hardware é o termo em inglês empregado para designar todo e qualquer componente, parte e sistema, capaz de realizar um processamento computacional, isto é, um processamento de modificação e controle de dados numéricos. Exemplos de hardware são o computador, suas partes, componentes e periféricos (monitor de vídeo, disco magnético, impressora, etc). Um computador é constituído pelas seguintes unidades:

- **unidade central de processamento (CPU - *Central Processing Unit*)**: responsável pela execução de instruções e pelo controle da operação de todas as unidades do computador.
- **unidade de armazenamento** de instruções e dados, que pode ser dividida em unidade primária, para armazenamento em tempo de execução (memória RAM - *Random Access Memory*) de curta duração e unidade de armazenamento secundária, de longa duração, é uma memória permanente constituída pela memória ROM (*Read-Only Memory*), pelos discos magnéticos (*floppy disk* e disco rígido) e pelos discos ópticos (CD-ROM, CD-RW) e magneto-ópticos.
- **unidades de entrada e saída (I/O - *Input/Output*)**, cuja função primária é a entrada e saída de dados do computador. Exemplos de dispositivos de entrada de dados são o teclado, o mouse, microfone e *joystick*, enquanto que dispositivos de saída típicos são o monitor de vídeo, caixa de som e impressora. Exemplos de periféricos que funcionam como dispositivos de entrada e saída de dados são a tela de vídeo sensível ao toque (*touch screen*) e o modem, usado para comunicação de dados entre computadores através de uma linha telefônica.

Atualmente, a capacidade dos computadores superam e muito as suas especificações e propósitos de uso original. Os computadores são capazes não apenas de armazenar, tratar e gerir

informações em quantidade e velocidade, são capazes também de proverem comunicação entre computadores e outros dispositivos eletrônicos digitais, tais como telefones, fax e televisores; são capazes de aceitar, manipular e apresentar informações na forma de voz, som, imagem, vídeo e texto; permitem o controle de outros dispositivos eletrônicos digitais, tais como semáforos, sistema de tráfego aéreo (radares, torre de controle, mesa de operação), sistema de comunicações (telefonia digital), sistemas bancários (caixa eletrônico, terminal de consulta, mesa de operação) e inúmeras outras aplicações essenciais para a vida cotidiana. A penetração da computação na vida diária se dá de tal forma, que aparelhos eletrodomésticos comuns, como torradeiras, máquina de fazer café são dotadas de um computador embutido num componente integrado miniaturizado e a tendência é que o computador de mesa que conhecemos hoje se torne um eletrodoméstico que vai comandar os outros aparelhos eletrodomésticos.

Os programas de computador são um conjunto de instruções que comandam o *hardware*. O *software*, por sua vez, designa um programa ou um conjunto de programas, capazes de atuar, modificar e controlar o processamento de dados lógicos e numéricos pelo computador. Existem três tipos de *software*:

- **sistemas operacionais e *firmware***: os sistemas operacionais são programas de computador que contém todas as instruções para o controle e a operação do computador. Exemplos de sistemas operacionais são o MS-DOS, Windows-9x (95, 98, ME e XP/Home), Windows NT, 200 e XP/Professional e o UNIX e suas variantes (Linux, FreeBSD, Solaris, Mac-OSX, etc) que "rodam" em diversas plataformas de *hardware*. A maioria dos sistemas operacionais provê uma interface de usuário gráfica (GUI - *Graphical User Interface*), de modo a facilitar a operação do computador sem a necessidade de memorização de comandos. O *firmware* é um conjunto de instruções que informa ao sistema operacional quais são os componentes de *hardware* que estão instalados no computador. Normalmente, o *firmware* é um conjunto de instruções que vem gravado numa memória ROM do tipo CMOS (*Complementary Metal Oxide Semiconductors*) instalada na placa-mãe. Em microcomputadores do tipo PC o *firmware* também é chamado de BIOS (*Basic Input-Output System*) a que muitos técnicos se referem como CMOS (por causa do tipo de memória).

- **linguagens de programação**: são as ferramentas para a construção de *softwares*, tanto para sistema operacional como para aplicações. Todos os programas que rodam num computador são feitos à partir de uma linguagem de programação. Existem diversas linguagens de programação, incluindo os seus dialetos que, geralmente, são constituídos por extensões da linguagem feitos por um fabricante de software em particular. Exemplos de linguagens de programação são: FORTRAN (linguagem de uso científico), COBOL (linguagem de uso comercial), BASIC, Pascal, C, C++ e Java. Existem linguagens de programação implementadas dentro de um software de aplicação e que são denominados *scripts*, como o VisualBasic for Applications (VBA), que é a linguagem *script* encontrada nos programas de processamento de texto Word, planilha Excel e banco de dados Access, todos integrantes do pacote de *software* Office da Microsoft. Outros programas que, originalmente foram criados como programas de aplicação com recursos de programação *script*, como os *softwares* de gerenciamento de banco de dados, evoluíram para linguagens de programação de banco de dados, como é o caso da linguagem SQL, desenvolvida pela IBM e pelo programa Oracle da empresa homônima.

- ***software* aplicativo**: programas de computador desenvolvidos para o usuário final, podem ser classificados como *software* de aplicação. Os *softwares* de aplicação geralmente são programas desenvolvidos para uma aplicação específica como, por exemplo, um *software* de controle de contas a pagar e receber ou um *software* de planilha eletrônica ou de processamento de texto. Na medida em que o *hardware* foi evoluindo (processadores mais velozes, memórias e discos com maior capacidade de armazenamento, etc), os programas aplicativos foram evoluindo englobando diversas tarefas e agregando outros programas num "pacote", como o MS Office.

Na Fig. 1.1 é esquematizado, em nível hierárquico, a relação entre hardware, software e o usuário (ser humano). Quanto mais inferior o nível, mais ele se aproxima do nível puramente físico em que enxergamos um computador como sendo um conjunto de componentes eletrônicos, placas de circuito e gabinetes, sem uma função lógica (e inteligente) a fazê-lo funcionar. À medida que subimos nos diversos níveis, aproxima-nos do nível puramente lógico, representado pela inteligência criadora do computador, o ser humano. Neste nível, estamos numa camada mais abstrata em que os conceitos são baseados na lógica e no raciocínio para criarmos os programas que irão interagir com o nível físico. Um programa de computador é, na essência, um conjunto de instruções transcritas para a linguagem do computador da inteligência (abstrata) do seu criador. Usualmente, ele é confundido pelo disquete no qual é armazenado, mas na realidade trata-se de uma entidade lógica relacionada com a capacidade intelectual do seu autor em descrever de forma algoritmica a sequência para a consecução de uma atividade executada pelo computador.

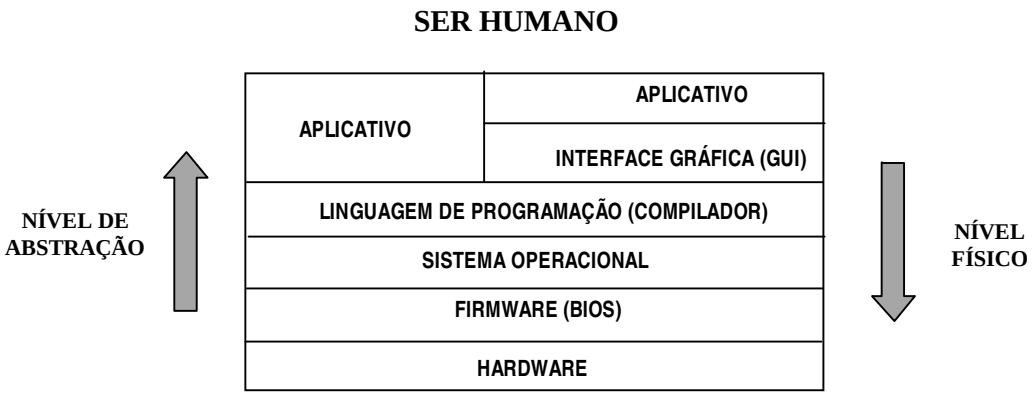



Fig. 1.1 - Modelo hierárquico para um computador.

## Arquitetura de Microcomputador

Um computador é essencialmente uma máquina de processamento de dados. Ele recebe dados ou requisição de informações, processa-os e fornece as informações ou dados requisitados de modo ordenado, digerido e reduzido, em forma de tabelas, gráficos, imagens, texto, som, etc.

Um microcomputador é um tipo de computador no qual a unidade central de processamento (CPU) é constituída por um circuito integrado de uso genérico de ultra alta escala de integração (ULSI - *Ultra Large Scale of Integration*) denominado microprocessador. Devido à sua disponibilidade, o microcomputador vem encontrando inúmeras aplicações em diferentes

áreas, como na comunicação de dados, em redes de computadores, como sistema de aquisição de dados e de controle de instrumentação nas áreas científica, médica e industrial, como *videogame* e centro de entretenimento. Internamente, um microcomputador organiza-se da forma esquematizada na Fig. 1.2.

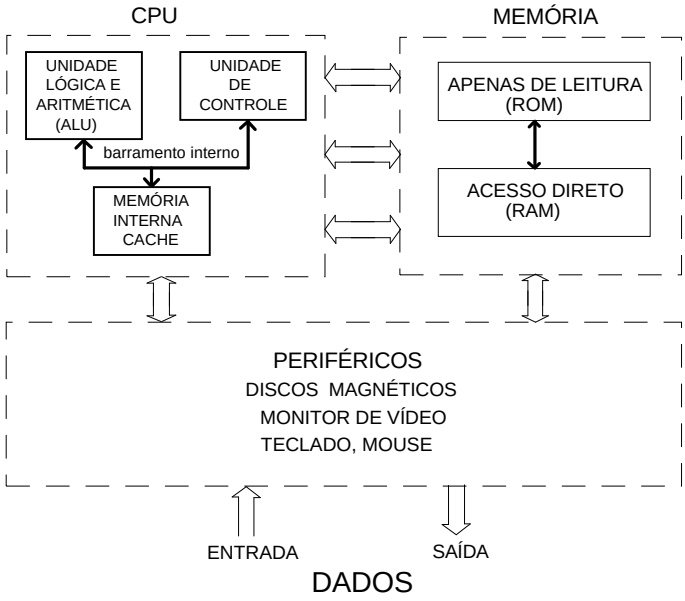



Fig. 1.2 - Arquitetura de um microcomputador.

Um microprocessador é um circuito integrado de elevadíssimo grau de integração, contendo milhões de transistores, é constituído basicamente por três unidades:

i. Unidade de Controle: obtém as informações que estão armazenadas na memória principal, interpreta-as e dá sequência às operações para as outras unidades para executar as instruções;
ii. Unidade Lógica e Aritmética: unidade que realiza as operações lógicas e aritméticas. As operações lógicas principais são a multiplicação lógica (AND lógico), adição lógica (OR lógico), negação lógica, inversão ou complementação lógica (NOT lógico), além de outras operações como NAND, NOR, XOR, etc. As operações aritméticas são a adição, subtração, multiplicação, divisão e deslocamento.
iii. Memória interna cache: realiza operação de armazenamento da parcela de dados da memória principal mais requisitadas, com a finalidade de aumentar a velocidade de acesso aos dados entre a CPU e a memória principal.

Outras unidades podem ser agregadas na pastilha do circuito integrado para aumentar a velocidade de processamento e melhorar o desempenho do processador.

A Fig. 1.3 mostra a arquitetura do microprocessador Pentium da Intel, com tamanho típico de circuito de 0,6 μm e contendo mais de três milhões de transistores. Observar que o processador de operações flutuante (*Pipelined floating point*) e o processador de operações inteiras (*Superscalar integer execution units*) são unidades adicionais, que tem como justificativa a capacidade de processamento superescalar e vetorial, características essas de supercomputadores, bem como suporte a unidade de processamento de operações com números complexos (*Complex instruction support*) que melhoram o desempenho do computador na execução de diversas tarefas simultaneamente (processamento paralelo multitarefa) e no

processamento numérico intensivo (como na geração de gráficos 3-D, execução de sons no formato MP3 e exibição de vídeo digital).

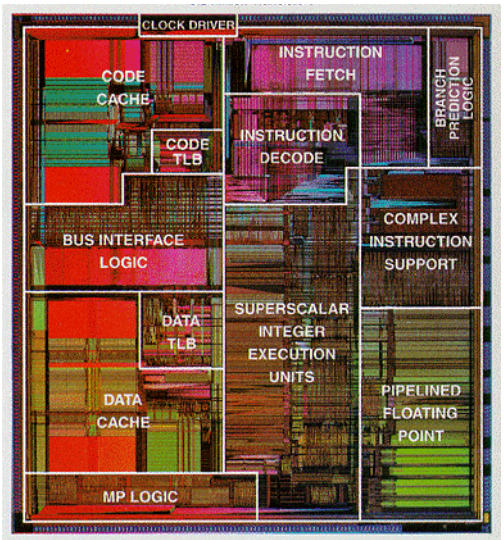



Fig. 1.3 - Esquema do circuito do microprocessador Pentium.

A memória é usada para armazenar instruções e dados operados pela CPU. Existem dois tipos de memória: memória ROM e memória RAM. A memória ROM ou memória apenas de leitura (*Read-Only Memory*) armazena principalmente as informações que necessitam ficar armazenadas permanentemente, como aquelas relativas ao *hardware* (tipo e quantidade de discos magnéticos, tipo de controladora de vídeo, endereçamento e quantidade de memória). A memória RAM ou memória de acesso direto (*Random-Access Memory*) é um tipo de memória volátil, isto é, as informações armazenadas nela são temporárias e se perdem quando o computador é desligado. A memória RAM é utilizada principalmente para armazenar dados e instruções relativos aos programas a serem executados e que ficam disponíveis apenas durante o tempo de execução.

## O Computador Digital

Desde os primórdios da Computação, nos anos 40, até os dias de hoje, os computadores vêm sofrendo um contínuo processo de desenvolvimento. Entretanto, o princípio fundamental de operação do computador não mudou, desde o ENIAC em 1945 e o EDVAC em 1952, que foi o primeiro computador integralmente eletrônico. Os computadores atuais são digitais (isto é, processam as informações utilizando números binários) que processam os dados e as instruções na CPU, com armazenamento na memória. Este modelo computacional deve-se ao matemático John Von Neumann, que estabeleceu os princípios dos computadores atuais e que por isso também são chamados de “computadores Von Neumann”. Pelo fato de operarem no formato numérico binário, significa que os números de base decimal a que estamos familiarizados devem ser convertidos no seu correspondente binário. Da mesma forma, o alfabeto e os símbolos gráficos (!?,.%$#<> etc) também devem ser convertidos em seu equivalente codificado em binário.

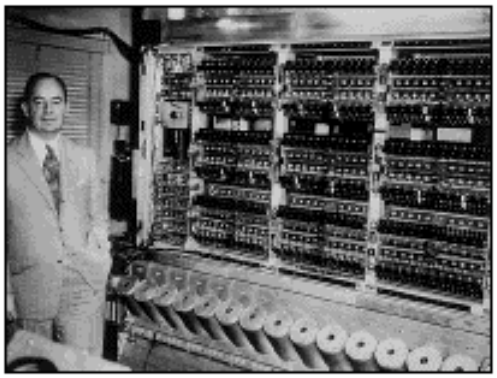


Fig. 1.4 - John Von Neumann e o computador EDVAC, o primeiro computador digital do mundo.

A quantificação da informação armazenada e processada por um computador é feita através do **byte** (simbolizado pela letra B maiúscula), que é igual a 8 **bits** (simbolizado pela letra b minúscula). Em termos aproximados, um byte é equivalente à um caracter, e a informação é quantificada em termos de múltiplos de bytes, que são potências de 2, como veremos adiante, e estão apresentados como ordens de grandeza do byte, como descrito a seguir:

1 kB = $2^{10}$ bytes = 1.024 bytes = 8.192 bits = 8 kb

1 MB = $2^{20}$ bytes = 1.048.576 bytes = 1.024 kB

1 GB = $2^{30}$ bytes = 1.073.741.824 bytes = 1.048.576 kB = 1.024 MB

Assim, um computador que contenha uma unidade de disco magnético de 650 MB de capacidade, é capaz de armazenar 650 x 1.048.576 = 681.574.400 bytes de informação, ou o equivalente a aproximadamente 681 milhões de caracteres, ou o equivalente a 180 mil páginas ou a cerca de 400 volumes de livros ou o equivalente a 40 volumes da Enciclopédia Britannica contendo somente texto. Para efeito de comparação, 650 MB também é a capacidade de armazenamento de um CD-ROM. O equivalente em CD a um arquivo de som digital é cerca de 75 minutos de gravação e equivalente à 100 imagens fotográficas coloridas de média definição (600 dpi - dpi = pontos por polegada).

## Linguagens de Computador

No início da Computação, a programação era realizada através da abertura e fechamento de válvulas eletrônicas por meio de chaves que controlavam a passagem de corrente pelas válvulas. Era uma tarefa essencialmente de manipulação física do *hardware* (Fig. 1.5).

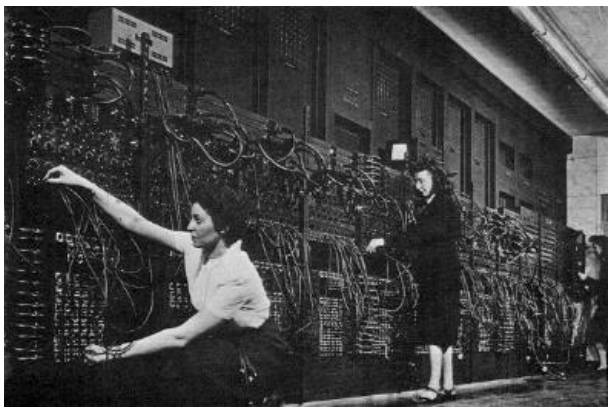


Fig. 1.5 - "Programando" o ENIAC.

À partir dos anos 50, com o desenvolvimento das linguagens de programação, que iniciou-se primeiramente com as linguagens de máquina (baseadas em codificação por números binários, 0 e 1), até as linguagens mais naturais como o FORTRAN e o COBOL, a programação de computadores vêm se afastando do nível físico e se torna cada vez mais uma atividade "abstrata" em que um algoritmo escrito em linguagem mais próxima à humana (daí o nome, linguagem "natural") tornando a programação à codificação de uma seqüência de instruções baseada numa linguagem de descrição de comandos. As primeiras linguagens foram projetadas para realização de tarefas específicas e foram evoluindo para linguagens de uso geral e hoje, se observa novamente uma tendência de dispor-se de linguagens de uso específico.

À seguir, listamos as principais linguagens de programação.

**FORTRAN**

Uma das mais antigas linguagens de programação, o FORTRAN que é um siglônimo de *FORmula TRANslator*, foi projetado para ser uma linguagem para manipulação de operações matemáticas, originalmente em computadores *mainframe*. Na época em que foi criado, a programação era feita em cartões perfurados, um para cada instrução do programa, característica essa que influenciou numa série de comandos que permanecem como herança até hoje. Embora seja uma das linguagens mais antigas, o FORTRAN evoluiu juntamente com a informática e hoje, a versão padronizada mais recente, o FORTRAN-90 possui uma série de recursos disponíveis nas linguagens mais modernas e mantém uma grande comunidade de programadores em todo o mundo ainda desenvolvendo programas científicos e de engenharia altamente complexos e sofisticados. Por essas características, ainda o FORTRAN é a *lingua franca* de inúmeros cientistas e engenheiros.

**COBOL**

O *COmmon Business Oriented Language* ou COBOL é a linguagem desenvolvida na mesma época do FORTRAN para criar programas de aplicação comercial, que envolvem a criação e manipulação de informações comerciais disponíveis em banco de dados, usando uma linguagem de comandos em inglês. Devido ao fato de ser uma linguagem com capacidade de manipulação de registros de dados comerciais, a sua capacidade de manipulação matemática é limitada às operações aritméticas básicas.

**Pascal**

A linguagem Pascal deve esse nome ao filósofo, matemático e físico francês Blaise Pascal, a quem Niklaus Wirth, professor do Instituto Técnico Federal (ETH) da Suíça, criador dessa linguagem homenageou. O Pascal foi projetado como uma linguagem de ensino de programação, daí a sua ampla aceitação em círculos acadêmicos de todo o mundo como a primeira linguagem a ser ensinada em cursos de programação para estudantes de ciências exatas. A sua construção força aos alunos à aprenderem e programar de forma estruturada e modular. Outra vantagem é que o Pacal foi desenvolvido para ser independente da plataforma de *hardware* e do sistema operacional.Assim, um programa escrito num computador poderia ser compilado sem modificação em outro tipo de computador, com diferente processador e sistema operacional.

**BASIC**

A linguagem BASIC, que é o acrônimo de *Beginner's All-purpose Symbolic Instruction Code*, tal como a linguagem Pascal, também foi criada para o ensino de programação de computadores em nível introdutório no Darthmouth College, EUA. No início, o BASIC foi a primeira linguagem interpretada disponível para uso geral e uma das primeiras a serem disponíveis pela então recém-criada empresa produtora de *software* Microsoft nos primeiros microcomputadores fabricados na década de 1970. A rápida popularização dos microcomputadores nos anos seguintes também popularizaram a linguagem BASIC entre os jovens aficionados por jogos eletrônicos em computador, resultando numa geração de programadores que aprendeu o BASIC como primeira linguagem de programação. Atualmente, a linguagem Microsoft Visual BASIC é uma das linguagens mais utilizadas na programação em ambiente Windows.

**Linguagem Assembly**

A linguagem Assembly é uma linguagem de representação simbólica da linguagem de máquina de um processador em particular, sendo por isso, considerada uma linguagem de baixo nível, isto é, de nível hierárquico próximo ao físico. Assim, cada processador tem sua linguagem assembly própria, apesar de que uma família de processadores, tais como a família Intel 80x86, pode compartilhar parte ou o todo de seu código assembly. Em português, as linguagens assembly são também denominadas linguagens montadoras (*assembler* em inglês).

**C**

Nos idos anos 60 e início dos 70 era comum os programadores criarem suas próprias linguagens de programação à partir de códigos assembler, por não haver a cultura de comercialização do *software* como um produto independente do *hardware* como hoje. Uma dessas linguagens experimentais foi denominada linguagem "A", que após alguns aperfeiçoamentos deu origem à linguagem "B". Esta linguagem, por sua vez, naturalmente evoluiu para o que hoje conhecemos como linguagem "C". O fato da linguagem ter evoluído e encontrado grande aceitação entre os programadores deriva do fato dela ser uma linguagem ao mesmo tempo simples e poderosa, capaz de programar o *hardware* em nível de linguagem de máquina ao mesmo tempo que possibilita uma construção sintática próxima à linguagem natural. Ela foi criada por Kerninghan e Ritchie, pesquisadores do Laboratório Bell dos Estados Unidos, como linguagem de desenvolvimento do sistema operacional Unix. Como este sistema foi adaptado para uma ampla variedade de plataformas de *hardware* quer foram adotados pela maioria dos fabricantes de computadores utilizados em aplicações críticas (bancos de dados, processadores de comunicação, gerenciadores de redes) a linguagem C ganhou grande popularidade entre os programadores de sistema e de aplicações sofisticadas.

**C++**

No final dos anos 80 com o aumento da capacidade do *hardware*, os desenvolvedores de *software* não conseguiam acompanhar o ciclo de desenvolvimento com a mesma rapidez da evolução do *hardware*, ocasionando o que se chamou de crise do *software*. Para fazer frente ao aumento da complexidade na criação dos *softwares*, os cientistas da computação elaboraram os conceitos de *objetos* e de *programação orientada a objects* , que determinaram

um novo paradigma de programação que as linguagens de programação da época, como a linguagem C padrão (o chamado ANSI-C), não tinham suporte à estrutura lógica desse novo paradigma. Devido ao fato da linguagem C ser (e ainda é) a linguagem de desenvolvimento mais utilizada pelos programadores profissionais, um tipo de extensão da linguagem proposto por Bjarne Stroustrup do mesmo Laboratório Bell onde foi criado a linguagem C foi desenvolvido. Esta linguagem denominada C++ é retro-compatível com programas escritos em linguagem C, ao mesmo tempo que adota uma estrutura para o desenvolvimento de programas de acordo com o paradigma de orientação a objetos.

**Linguagens Scripting**

As linguagens *scripting* existem desde os anos 60. Entretanto, o poder e sofisticação dessas linguagens aumentou dramaticamente nos últimos anos. Combinando com o avanço na tecnologia de *hardware*, o uso de linguagens *scripting* ampliou o horizonte de aplicações para um sem número de tarefas. As linguagens *scripting* foram criadas para funcionar como "cola" na integração de componentes de *software* e aplicações criados em linguagens de programação convencionais, como as descritas anteriormente. Elas avançaram no vácuo da crise de *software* dos anos 80 e funcionaram e têm funcionado como uma mola propulsora na criação de aplicativos gráficos e distribuídos, que hoje mantém em funcionamento parcela significativa dos *sites* comerciais da internet. À seguir, relacionamos algumas linguagens *scripting* mais utilizadas.

**Shell Unix (sh, csh, ksh, etc)**

O sistema operacional têm uma linguagem de comandos que possui a sintaxe da linguagem C e que possibilita a digitação de comandos interativamente. A estas linguagens *scripting* de linha de comando denomina-se *shell*. O primeiro *shell* do sistema operacional Unix foi criado no início dos anos 70 e uma série de outros *shells* (csh, ksh, bash, etc) foram criados para automatizar tarefas rotineiras de operação e administração do Unix. Um dos aspectos que tornam o Unix poderoso e singular é a sua capacidade de criar novas aplicações à partir da composição de diferentes aplicações já existentes e que, talvez seja uma das razões da popularidade do Unix como plataforma de desenvolvimento de *software* entre os desenvolvedores profissionais.

**Perl**

Criado por Larry Wall no final dos anos 80 com a finalidade de colocar em um único lugar as funções de muitas aplicações de processamento de texto Unix, tais como sh, sed e awk, o Perl (acrônimo de *Practical Extraction and Report Language*) rapidamente se tornou uma das ferramentas mais utilizadas pelos administradores de sistemas de informação. Com a chegada do WWW (internet com interface gráfica), o Perl adquiriu fama como linguagem para construção de *scripts* para páginas Web dinâmicas, isto é, que se atualizam com o acesso dos usuários.

**Tcl**

Criada por John Ousterhout no fim dos anos 80 como uma linguagem de comandos embutível para uso como ferramenta interativa. Completa-o a ferramenta de construção de interface

gráfica Tk. O Tcl e o Tk são disponíveis para todas as principais plataformas de *hardware* e *software*, sendo uma ferramenta essencial na criação de aplicativos multiplataforma. Hoje, a linguagem Tcl é usada para uma ampla variedade de uso, desde geração automática de conteúdo para internet, até gerenciamento de sistemas, passando por automação de projeto eletrônico.

**Visual Basic**

A linguagem da Microsoft pode ser considerada ao mesmo tempo como linguagem de programação de sistema e como linguagem *scripting*. Conforme mencionado anteriormente, ela se popularizou como ferramenta para criação rápida de aplicativos (em inglês, RAD *Rapid Application Development*), baseado na interface gráfica do Windows. A combinação do Visual Basic e da linguagem de componentes *scripting* VBX (atualmente denominado ActiveX) é a força motriz por detrás de inúmeros *sites* de comércio eletrônico da internet baseados na tecnologia de *software* ASP (acrônimo de *Active Server Pages*) que executa todas as tarefas de geração de páginas HTML dinâmicas e de administração de banco de dados. Esse sucesso em grande parte é devido à facilidade de integração propiciada pelo Visual Basic.

**Python**

O Python é uma linguagem orientada a objetos dinâmica, criada por Guido van Rossum no início dos anos 90 como uma proposta de facilitar o aprendizado de uma linguagem de programação e ao mesmo tempo prover uma ponte entre entre o *shell* e a linguagem C. É uma linguagem elegante, com uma sintaxe fácil de aprender, portável como o Tcl, com o qual compartilha a ferramenta GUI Tk e com uma extensa biblioteca de suporte matemático, gráfico e multimídia. É fácil de estender para diversas aplicações e ser embutido em C/C++, o que tem contribuído para a sua popularização. Originalmente projetada como uma linguagem *scripting* avançada, têm encontrado novos usos como ferramenta RAD para a Web, bem como para aplicações distribuídas.

**JavaScript**

Criada no meio da década de 90 pela Netscape para executar *scripts* embutidos em páginas HTML, como por exemplo, validação de preenchimento de formulários. O JavaScript se tornou um padrão *de facto* para *scripts* executados no lado do cliente. Embora não tenha relação com a linguagem Java, compartilha com a sintaxe dos principais funções e tipos de dados.

**Java**

O Java, originalmente desenvolvido por Bill Joy da SUN como uma linguagem de programação de dispositivos eletrônicos programáveis, encontrou o seu nicho de aplicação na Web como linguagem de geração de pseudo-códigos compilados orientados a byte e não a bits e independente de plataforma. Para ser executado, o Java pode rodar tanto do lado do cliente, quando o código é denominado *applet*, quanto do lado do servidor, quando é denominado *servlet* o programa compilado Java. Para ser executado no lado do cliente é

necessário que o navegador (*browser*) Web tenha instalado a máquina virtual Java (JVM - *Java Virtual Machine*) que é o interpretador de código orientado a byte do Java.

**PHP**

A linguagem PHP (um acrônimo recursivo para *PHP: Hypertext Preprocessor*) é uma linguagem *scripting* de uso geral, muito utilizada para o desenvolvimento de aplicações Web embútivel dentro de documentos HTML. Ela foi criada por Jamus Ledorf em 1995, inicialmente como simples *script* Perl para gerar estatísticas de acesso para seu currículo *online*. Ele nomeou esta série de *scripts* como *Personal Home Page Tools* (daí veio o nome PHP originalmente). Como mais funcionalidades foram requeridas, Rasmus escreveu uma implementação C muito maior, que era capaz de comunicar-se com base de dados, e possibilitava à usuários desenvolver simples aplicativos dinâmicos para Web. O que distingui o PHP de outras linguagens *script*, como Javascript é que embora o comando esteja escrito na página HTML, o código é executado no servidor. Se o *script* estiver no servidor, o cliente recebe os resultados da execução desse *script*, sem determinar como é o código fonte. Por causa dessa característica e também pela integração com programas *open source* como o MySQL (gerenciador de banco de dados) e Apache (servidor WWW), o PHP é muito utilizado na administração de *sites* com conteúdo dinâmico.

## Níveis de Linguagens de Programação

Existe apenas uma linguagem de programação que qualquer computador pode entender e executar: o seu código de máquina nativo. Este é o mais baixo nível em que se pode escrever um programa de computador. Todas as outras linguagens de programação são chamadas de linguagens de alto nível ou de baixo nível em função do seu grau de semelhança com a linguagem de máquina em termos de recurso e de sintaxe.

Assim, uma linguagem de baixo nível corresponde à linguagem de máquina, de modo que uma instrução nessa linguagem corresponde a uma instrução em código de máquina. Já uma instrução em linguagem de alto nível corresponde a uma série de instruções em linguagem de máquina. Em termos gerais, o número de instruções em linguagem de máquina equivalente a uma instrução em linguagem de alto nível é denominada ***pontos de função***.

As linguagens de baixo nível tem a vantagem de poderem ser escritas com instruções que acessam diretamente às particularidades da arquitetura da unidade central de processamento (CPU) tornando, assim, o programa extremamente eficiente, otimizando o uso tanto da memória RAM como do processador em sí. Entretanto, escrever um programa em linguagem de baixo nível demanda um tempo significativamente maior e requer um conhecimento mais aprofundado das características internas do processador, além do código estar mais sujeito a falhas plea maior complexidade de programação. Portanto, programação em linguagem de baixo nível é utilizada em programas pequenos ou em segmentos de código em linguagem de alto nível para otimizar o desempenho de partes críticas do programa.

As linguagens de alto nível permitem o desenvolvimento rápido de programas. Em comparação com os programas escritos em linguagem de baixo nível, o desempenho pode não ser tão bom, mas a economia de tempo na programação supera a menor eficiência na execução pelo fato do custo operacional mais elevado residir no desenvolvimento de *software* e não na *upgrade* do *hardware*. Como regra básica, o custo para escrever um programa é aproximadamente

constante para cada linha do código, independentemente da linguagem, de forma que escrever uma linha de instrução em programa escrito em linguagem de alto nível, que equivale a dez linhas de código escrito em linguagem de baixo nível, representa um custo de 1/10 do custo do programa escrito em linguagem de baixo nível.

## Representação de Números

Os números empregados no cálculo computacional podem ser de dois tipos: números inteiros e números em "ponto flutuante" (que representam os números reais da Matemática). Os computadores atuais representam os números internamente no formato binário, como seqüência de 0s e 1s. Apesar dessa representação ser conveniente para as máquinas, é anti-natural para os seres humanos, cujo sistema de numeração natural é o decimal. A seguir, apresentamos a representação de números decimais e binários e os métodos de conversão de um sistema a outro.

## Sistema de Numeração Decimal

O sistema de numeração decimal é o sistema natural, adotado em todas operações matemáticas cotidianas. Este sistema é de base 10, no qual todos os múltiplos e submúltiplos são expressos como potências de 10. Por exemplo, os seguintes números decimais podem ser escritos como uma combinação linear de potências de 10:

$$1537 = 1x10^3 + 5x10^2 + 3x10^1 + 7x10^0 \quad (1.1)$$

$$36{,}189 = 3x10^1 + 6x10^0 + 1x10^{-1} + 8x10^{-2} + 9x10^{-3} \quad (1.2)$$

$$6{,}032x10^{23} = 6x10^{23} + 0x10^{22} + 3x10^{21} + 2x10^{20} \quad (1.3)$$

Observar que a posição relativa de cada algarismo indica a potência pela qual ele está multiplicando. Assim, de acordo com esta convenção, o algarismo 3 do exemplo (1.1), que está na 2ª posição à contar da direita para a esquerda, está multiplicando $10^1$. O algarismo 7 (unidade) do mesmo exemplo é chamado de algarismo menos significativo e o algarismo 1 (milhar) o algarismo mais significativo. O algarismo 3 é o 2º algarismo significativo e o 5 o 3º algarismo significativo.

De acordo com os exemplos acima, qualquer número decimal pode ser escrito na forma geral como:

$$N = a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + ... + a_0 b^0 + a_{-1} b^{-1} + ... + a_{-m} b^{-m} \quad (1.4)$$

onde N é um dado número na base b e $a_n$, $a_{n-1}$, etc representam os coeficientes que multiplicam as correspondentes potências de b. Assim, $N = a_n a_{n-1}...a_0,a_{-1}\ a_{-2}...a_{-m}$ na representação usual (implícita). Detalhe importante a observar é o de os coeficientes que multiplicam potências de b cujos expoentes sejam $< 0$ estão separados por uma vírgula daqueles coeficientes cujos expoentes de b sejam $\geq 0$.

## Sistema de Numeração Binário

Atualmente, em que pese a nossa familiaridade com o sistema de numeração decimal, este tipo de representação numérica é inadequado para a representação da informação em computadores digitais. Os computadores digitais operam basicamente com dois tipos de sinais de tensão: alto e baixo. Matematicamente, pode-se expressar estes valores por 0 (baixo) e 1 (alto). À partir de um esquema de representação binária por valores de 0 e 1 podemos expressar qualquer quantidade numérica. Vejamos os seguintes exemplos de números decimais representados como potências de 2:

$$2{,}5_{10} = 1x2^{1} + 0x2^{0} + 1x2^{-1} \quad (1.5)$$

$$98{,}75_{10} = 1x2^{6} + 1x2^{5} + 0x2^{4} + 0x2^{3} + 0x2^{2} + 1x2^{1} + 0x2^{0} + 1x2^{-1} + 1x2^{-2} \quad (1.6)$$

Observar que o índice 10 em 2,5 e 98,75 indica que esses números são decimais.

Utilizando a fórmula de representação expressa pela equação (1), podemos reescrever os dois exemplos numéricos acima como:

$$2{,}5_{10} \equiv 10{,}1_{2}$$

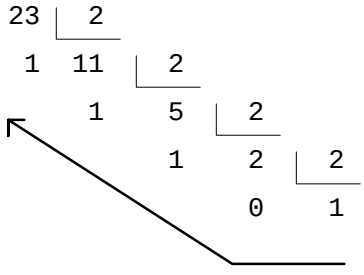
$$98{,}75_{10} \equiv 1100010{,}11_{2}$$

Como no caso dos números decimais, a posição relativa de cada algarismo binário indica a potência pela qual ele está multiplicando. Igualmente os coeficientes que multiplicam potências de 2 cujos expoentes sejam $< 0$ estão separados por uma vírgula daqueles coeficientes cujos expoentes de 2 sejam $\geq 0$.

## Conversão de Números Decimal-Binário

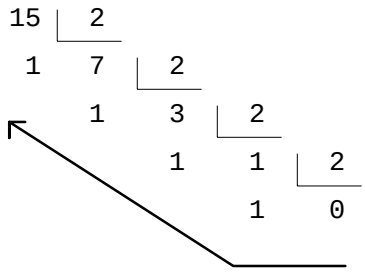
Para convertermos um número decimal para um número binário devemos aplicar um método para a parte inteira e outro para a parte fracionária. Para a parte inteira, aplicamos o método das divisões sucessivas, que consiste em se dividir o número decimal por 2, em seguida divide-se o quociente obtido por 2 e assim sucessivamente, até que o último quociente encontrado seja 1. O número binário inteiro será, então, formado pela concatenação do último quociente com os restos das divisões no sentido contrário ao que foram obtidos.

Por exemplo, vamos converter o número decimal 23:




$23_{10} = 10111_{2}$

Outro exemplo,

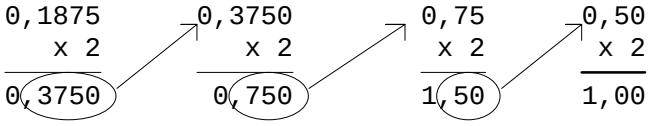



$15_{10} = 01111_2 = 1111_2$

Notar que neste exemplo necessitamos apenas de quatro dígitos binários para descrever $15_{10}$, pois o zero `a esquerda de $1111_2$ não é siginificativo.

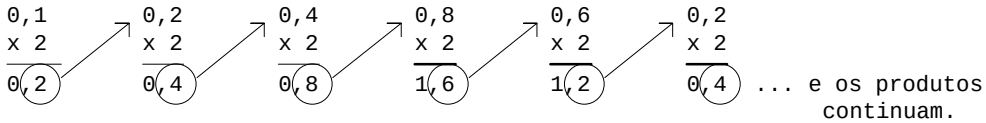
Para converter um número fracionário de base decimal para base binária aplicamos o método das multiplicações sucessivas. Ele consiste nas seguintes etapas: multiplicamos o número fracionário por 2; deste resultado, a parte inteira será o primeiro dígito do número fracionário binário e parte fracionária será novamente multiplicada por 2. O processo repete-se até que a parte fracionária do último produto seja zero.

Como exemplo, tomemos o número decimal 0,1875:




$0,1875_{10} = 0,0011_2$

Outro exemplo,




$0,1_{10} = 0,0001100110011..._2$

Como mostrado neste exemplo, nem sempre um número decimal exato possui uma representação binária exata. Este fato é a principal causa de erros de arredondamento no cálculo numérico em computadores digitais.

A conversão de bases para um número decimal que contém parcela inteira e parcela fracionária é constituída por duas partes distintas sobre as quais são aplicadas as respectivas regras de conversão:

$$23,1875_{10} = 10111,0011_2$$

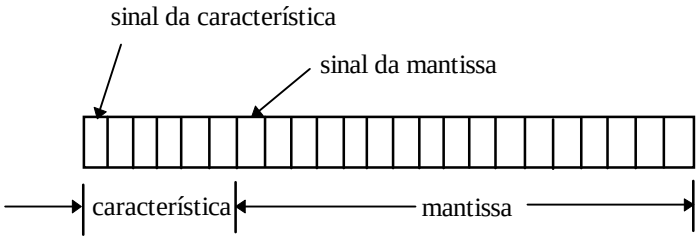
## Representação Binária em Ponto Flutuante

Em computação digital, um dígito binário é denominado bit (do inglês, ***bi**nary digi**t***). Um grupo de oito bits corresponde a 1 byte. Nos computadores digitais, a representação dos números binários é feita com um número finito de bits. A esse tamanho de bits é dado o nome de **palavra de computador**. O tamanho da palavra de computador depende de características internas à arquitetura do mesmo. Em geral, os microcomputadores padrão PC tem o tamanho de palavra de

16 e 32 bits. Computadores com arquitetura mais avançada possuem tamanho de palavra de 64 bits ou mais.

O tamanho de palavra de computador tem a seguinte implicação: quanto maior o tamanho da palavra, mais veloz e mais preciso será o computador. Para entender como o tamanho de palavra afeta a precisão e, consequentemente, os erros de arredondamento do cálculo computacional, veremos a representação de números reais binários em notação de ponto flutuante.

Um número binário em ponto flutuante (número real) é representado da seguinte forma:




Pela norma IEEE-754 do Instituto dos Engenheiros Elétricos e Eletrônicos (IEEE), o dígito mais à esquerda (também chamado de dígito mais significativo - em inglês, MSB) é o sinal da característica: 0 (positivo) e 1 (negativo). Os dígitos seguintes representam o valor binário da característica seguido pela mantissa. O número de bits da característica é obtido pela diferença entre o tamanho da palavra de computador e o número de bits da mantissa que se seguem à característica, sendo que o primeiro bit da mantissa representa o seu sinal. A convenção para o sinal da mantissa é o mesmo da característica, isto é, 0 é positivo e 1, negativo.

Os números em ponto flutuante no formato IEEE tem a seguinte representação:

| | Tamanho (bytes) | Nº de bits de sinal | Nº de bits do expoente | Nº de bits da mantissa | "Bias" |
|---|---|---|---|---|---|
| Precisão simples | 4 | 1 | 8 | 23 | 127 |
| Dupla precisão | 8 | 1 | 11 | 52 | 1023 |

- ***Bit de sinal***

  O bit de sinal apresenta zero como número positivo e um como número negativo.

- ***Expoente (característica)***

  O campo do expoente precisa representar tanto números positivos como números negativos. Para fazer isto, um valor "bias" é adicionado ao expoente para obter o expoente armazenado no computador. Para a norma IEEE o valor é de 127 para variável em precisão simples. Assim, um expoente significa que 127 é armazenado no campo do expoente. Um valor armazenado de 200 indica um expoente real de (200 – 127) ou 73. Expoentes armazenados de –127 e +128 são usados para fins especiais.

- ***Mantissa***

  A mantissa representa os bits de precisão de um número. Qualquer número pode ser expresso em notação científica. Por exemplo, o número cinco pode ser representado em qualquer uma das seguintes formas:

$5.00 \times 10^{0}$
$0.05 \times 10^{2}$
$5000 \times 10^{-3}$

Por conveniência, os números em ponto flutuante são armazenados na forma *normalizada*. Em notação científica, a forma normalizada de representar o número cinco é como $5.0 \times 10^{0}$, enquanto que na notação de ponto flutuante, a parte inteira não é representada, de modo que o número cinco é representado na forma *normalizada* como $0.5 \times 10^{1}$.

## Tipos de Dados Numéricos no Computador

Os tipos de dados numéricos no computador representados na linguagem C são equivalentes aos tipos numéricos de outras linguagens, como o FORTRAN, o BASIC e o Pascal.

**Tipo INTEIRO**

Representa os números inteiros, empregando 1, 2 ou 4 bytes, de acordo com o tamanho do número que desejamos expressar.

| Tipo | Denominação | bytes | Faixa | Algarismos significativos |
|---|---|---|---|---|
| `short` | inteiro curto | 1 | -128 a 127 | 3 |
| `int` | inteiro | 2 | -32.768 a 32.767 | 5 |
| `long` | inteiro longo | 4 | -2.147.483.648 a 2.147.483.647 | 10 |

**Tipo REAL**

Os números fracionários ou decimais são representados como números reais, cuja notação empregada é a de número com ponto decimal fixo ou ponto flutuante. A vantagem deste último é permitir expressar números muito pequenos ou números muito grandes de forma automática. Existem dois tipos de números reais de acordo com a sua precisão: 1) Precisão simples, com 4 bytes e 2) Dupla precisão, com oito bytes. A tabela seguinte apresenta os tipos representativo dos números reais.

| Tipo | Denominação | bytes | Faixas de valores | Algarismos significativos |
|---|---|---|---|---|
| `float` | Precisão simples | 4 | $-3{,}4028235.10^{38}$ a $-1{,}1754944.10^{-38}$<br>$1{,}1754944.10^{-38}$ a $3{,}4028235.10^{38}$ | 7 |
| `double` | Dupla precisão | 8 | $-1{,}797693134862316.10^{308}$ a $-2{,}225073858507201.10^{-308}$<br>$2{,}225073858507201.10^{-308}$ a $1{,}797693134862316.10^{308}$ | 15 |

# Aritmética Computacional e Erros Computacionais

Quando realizamos cálculo manualmente, os erros de arredondamento da calculadora são desprezíveis, porque a quantidade de cálculo que podemos operar é pequeno. No computador, geralmente, a quantidade de operações aritméticas que se pode realizar é muito maior do que aquelas realizadas manualmente, de forma que o erro de arredondamento do dispositivo de cálculo se torna importante. Outro problema é que no cálculo manual temos controle da propagação do erro porque, visualmente, estamos conferindo o resultado de cada operação aritmética ao digitá-lo na calculadora. No computador não temos como checar cada operação, tendo em vista a velocidade com elas são realizadas e também pela quantidade, que impossibilita a conferência dos resultados das operações aritméticas. Desta forma, a verificação dos resultados e o controle sobre a propagação de erros em programas de computador é essencial para se atingir resultados consistentes e confiáveis.

As seguintes considerações tem que ser levadas em conta ao se realizar cálculos numéricos no computador:

- A aritmética computacional não é a mesma coisa que a aritmética à base de lápis e papel. No cálculo manual é sempre possível monitorar os resultados intermediários e ajustar a precisão dos cálculos. Na aritmética computacional, cada número tem uma quantidade de algarismos fixas que muitas vezes podem ser inadequadas para o cálculo repetitivo;
- O cálculo manual usualmente é realizado para um pequeno número de operações aritméticas, enquanto que o cálculo computacional pode envolver bilhões de operações aritméticas. Assim, pequenos erros que poderiam passar despercebidos no cálculo manual, podem arruinar completamente o resultado do cálculo computacional, por causa da acumulação e propagação de erro.

**Imprecisão intrínseca da representação binária em ponto flutuante**

Todo o número inteiro decimal pode ser representado exatamente por um número inteiro binário; porém, isto não é verdadeiro para números fracionários. Na realidade, todo o número decimal irracional também será irracional no sistema binário. No sistema binário, apenas números racionais que podem ser representados na forma p/q, no qual q é uma potência inteira de 2, podem ser expressos exatamente com um número de bits finito. Mesmo frações decimais comuns, tal como o número decimal 0,0001 não podem ser representados exatamente em binário (0,0001 é uma fração binária repetitiva com período de 104 bits!).

Um programa simples de soma de números decimais fracionários, como o seguinte:

```
void main()
{
   int i;
   float soma = 0.;
   for(i=1;i<=10000;i++)
      soma = soma + .0001;
   printf("Soma = %10.7f", soma);
}
```

apresentará o número `1.0000535` como saída ao invés do número exato `1`. O pequeno erro de representação do número decimal 0,0001 em binário se propagará pela soma, comprometendo o resultado da soma final.

**Definição de erro absoluto e erro relativo**

- **Erro absoluto**

$$\text{erro absoluto} = |\text{valor verdadeiro} - \text{valor aproximado}|$$

$$e_x = \Delta x = |x_{exato} - x_{aprox}|$$

Podemos representar, matematicamente, $x_{aprox} = \bar{x}$ e $x_{verdadeiro} = x$, de modo que

$$\Delta x = |x - \bar{x}|$$

- **Erro relativo**

O erro relativo é o erro absoluto dividido pelo valor verdadeiro:

$$\text{erro relativo} = \frac{\Delta x}{x} = \frac{|x - \bar{x}|}{x}$$

e que, frequentemente, é expresso também como erro percentual, multiplicando-se o erro relativo por 100:

$$\text{erro percentual} = \text{erro relativo} \times 100$$

# Tipos de Erros

- **Erro de arredondamento**

Os erros de arredondamento são causados pela limitação dos dispositivos empregados no cálculo numérico como, por exemplo, uma régua de cálculo que possui uma limitação geométrica, uma calculadora eletrônica com número de dígitos limitado no "display" ou mesmo um computador com o erro de representação de um número decimal no seu equivalente binário.

Por exemplo, consideremos o número decimal exato $0,1_{10}$, cujo equivalente binário é representado pela dízima $Ø.\ ØØØ11ØØ11ØØ11_2$ ... Para um computador com tamanho de palavra de 16 bits, o número decimal $0,1_{10}$ é armazenado como:

$$0,1_{10} \approx Ø.\ ØØØ11ØØ11ØØ11ØØ1_2 = 0,099990844_{10}$$

de modo que o erro de arredondamento é dado por:

$$\text{erro absoluto} = |0,1 - 0,099990844| = 0,000009155 \approx 9.10^{-6}$$

- **Erro por estouro de memória ("Overflow") e "Underflow"**

A variável numérica real em precisão simples (*default*) pode conter no máximo o número $10^{38}$ e no mínimo $10^{-38}$, como visto anteriormente. O erro de "overflow" ocorre quando o resultado de uma operação aritmética excede o valor de $3{,}4028235.10^{38}$. Analogamente, o erro de "underflow" ocorre para uma variável real em precisão inferior a $1{,}1754944.10^{-38}$.

A multiplicação e a divisão podem acarretar erro de "overflow" como de "underflow". O caso mais extremo de "overflow" na divisão acontece quando ocorre uma divisão por zero, mas, geralmente, tal tentativa provoca uma interrupção na execução do programa com a emissão de um aviso pelo programa de "overflow".

- **Erro de Truncamento**

O erro de truncamento é um erro devido ao método de aproximação empregado para o cálculo de uma função exata. Por exemplo, considere a expansão da função exponencial em séries de potência da forma:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Por se tratar de uma série infinita, devemos escolher um número de termos limitado da série para que possamos computar o valor numérico da função $e^x$. Escolhemos aproximar a série infinita por uma série contendo três termos, isto é, pela aproximação:

$$e^x \approx 1 + x + \frac{x^2}{2!}$$

Não importa a quantidade de algarismos significativos que utilizemos no cálculo de $e^x$ pela série "truncada", o resultado será sempre aproximado e, portanto, sempre terá um erro, que é denominado **erro de truncamento**.

**Exemplo**
Calcule o valor numérico de $e^1$ = e (número de Euler) empregando a série truncada de 2ª ordem:

$$e^1 \approx 1 + 1 + \frac{1^2}{2!} = 1 + 1 + 0{,}5 = 2{,}5$$

Sabendo-se que o valor exato do número de Euler com quatro algarismo significativos é igual a 2,718, podemos avaliar o erro de truncamento como:

$$\text{erro absoluto} = |2{,}718 - 2{,}500| = 0{,}218$$

ou, em termos do erro percentual, como:

$$\text{erro} = (0{,}218/2{,}718) \times 100 = 8{,}0\%$$

# Referências Bibliográficas *Online*

Bjarne Stroustrup FAQ (*Frequently asked questions*)
http://www.research.att.com/~bs/bs_faq.html

Tcl website
http://dev.scriptics.com/doc/scripting.html

História do Perl
http://www.perl.org/press/history.html

Python language
http://www.python.org

A história do PHP e projetos relacionados
http://www.php.net/manual/pt_BR/history.php

Java
http://java.sun.com/

Notação de números em ponto flutuante IEEE Standard 754
http://research.microsoft.com/~hollasch/cgindex/coding/ieeefloat.html

Desastres causados por erro de cálculo numérico
http://www.ima.umn.edu/~arnold/disasters/