UnB Departamento de Ciência da Computação

2016/2

Organização do Computador

Prof. Guilherme N. Ramos

Considerando que a *Ciência da Computação* estuda a fundamentação teórica das construções computacionais, bem como suas aplicações em dispositivos tecnológicos e sistemas de computação [1], percebe-se que está intrinsecamente associada a uma ferramenta: o computador. De forma simplificada, um computador:

1. realiza cálculos [simples] rapidamente;
2. lembra de [muitos] resultados.

Os cálculos são a execução de operações primitivas (e qualquer um pode criar suas próprias operações); já os resultados são informações resultantes de operações. Cada execução tem um custo [de tempo], cada resultado tem um custo [de espaço].

"*Computadores não resolvem problemas, eles executam soluções.*"

Laurent Gasser

1 Computação

Por exemplo, supondo um livro de 1000 páginas em que cada página tenha 1000 palavras, qual seria o custo de alguém procurar uma palavra em uma página? Se consigo olhar duas palavras por segundo, então conseguiria avaliar todas as palavras de uma página em 500 segundos ($\approx$ 8 minutos). E para procurar uma palavra no livro? Bastaria repetir o processo para cada página (1000$\times$), terminaria em quase 6 dias. É importante saber quanto tempo para me planejar direito...

Considerando um jogo de xadrez, em média existem 35 movimentos possíveis por jogada. Supondo que eu considere uma a cada 30 segundos, demoraria cerca de 17 minutos para me decidir (uma partida tem cerca de 80 jogadas, levaria $\approx$ 23 horas). Mas e se eu quisesse considerar todas as possibilidades para 3 jogadas a frente? Cada jogada levaria $(17)^3$ minutos ($\approx$ 37 anos). Talvez seja melhor considerar um jogo mais simples.

Supondo que seja necessária uma representação do tabuleiro para a análise que definirá qual seria a jogada certa. Se cada tabuleiro fosse armazenado em 512 bytes de memória, os 35 tabuleiros necessários para uma jogada ocupariam $\approx$ 18 kB. Considerando três jogadas, seriam necessários $\approx 5,8$ TB de memória para armazenar todas as possibilidades...

Além destas considerações sobre custos de tempo e espaço, há outras questões teóricas de interesse. Por exemplo, é possível descobrir se um programa termina sua execução? Suponha um programa A executa uma instrução simples, e um programa B que executa a bizarra instrução `Não Termine`, que força o computador a realizar uma operação interminável (por exemplo, contar todos os números inteiros).

Programa A

```
1 Escreva("Terminei.")
```

Programa B

```
1 Não Termine
```

Ao executar A, o programa escreve "Terminei." e pára sua execução. Ao executar o programa B, ele simplesmente não termina. Agora imagine um programa C que consegue avaliar um outro programa X qualquer (portanto X é a entrada de C), e determinar se ele termina ou não:

Programa C

```
1 Se Programa X Termina
2     Escreva("O programa termina.")
3 Senão
4     Escreva("O programa não termina.")
```

Programa D

```
1 Se Programa X Termina
2     Não Termine
3 Senão
4     Escreva("Terminei.")
```

Se dermos o programa A como entrada para C testar, C mostra "O programa termina.". Analogamente, caso B seja fornecido a C, este mostra "O programa não termina.". O programa D é uma variação de C. Se dermos o programa A como entrada para D testar, ele verifica que A pára, portanto D não termina sua própria execução. Da mesma forma, caso B seja fornecido como entrada a D, este último verifica que B não pára sua execução, mostra "Terminei." e termina sua própria execução.

D, assim como A, B, e C, também é um programa e, portanto poderia ser avaliado por ele mesmo. Considere duas cópias do mesmo programa, D_A e D_B, que são o programa D avaliando os programas A e B, respectivamente. Ao executar novamente o programa D com D_A como entrada, sabe-se que D_A não termina e, portanto, o programa D que o avalia mostra "Terminei." e termina sua própria execução. Como D e D_A são o mesmo programa, o resultado é que D termina e também não termina. Ao executar o programa D com D_B como entrada, sabe-se que D_B termina e, portanto, o programa D que o avalia não pára sua própria execução. Como D e D_B são o mesmo programa, o resultado é que D não termina e também termina.

Estas considerações são relevantes porque, na prática, é preciso saber se é possível resolver o problema e, em sendo, qual o esforço necessário para obter a solução. Por exemplo, é possível analisar todas as possibilidades de um jogo de xadrez, mas ninguém se interessa em aguardar dias/anos pelo resultado. Também é possível calcular a velocidade de aproximação da pista e o ângulo de pouso de um avião, mas ninguém se interessa por este resultado se não for obtido antes da aterrissagem.

2 Resolução de Problemas com Computador

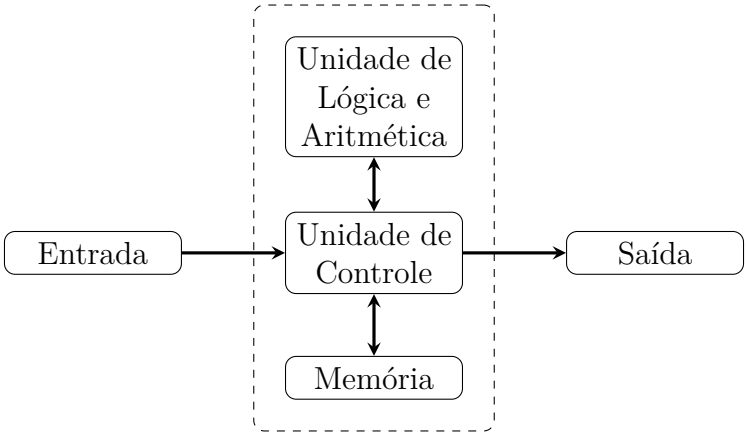
Para resolver problemas com o computador, primeiro é preciso perguntar: o que é *computação*? Para responder, é preciso saber o que é conhecimento? Para tanto, basta lembrar os tipos de conhecimento *descritivo* (o que?) e, principalmente, *procedural* (como?).

A descrição precisa de um procedimento para realizar uma tarefa, de modo que um computador possa executá-la, é chamada algoritmo computacional [2], então - para responder a pergunta inicial - basta saber como transformar estas instruções em um processo mecânico que pode ser realizado pelo computador? Uma solução simples é construir uma máquina que "sabe" como realizar apenas uma tarefa específica, como: uma calculadora (para operações matemáticas), o Computador de Atanasoff–Berry (para resolução de equações lineares), a Bomba de Turing (para decodificação de mensagens da Enigma), etc.

Isto implica em ter uma máquina específica para cada problema que se deseja resolver, uma situação custosa e inconveniente. E se houvesse uma máquina que armazena e interpreta instruções? Um *Computador de "Programa Armazenado"*?

Este computador armazena uma sequência de instruções (programa) que são construídas a partir de suas operações primitivas de aritmética e lógica, testes simples, e manipulação de dados. Um programa seria a composição correta de instruções utilizando estas primitivas, que fica armazenado

na memória. A máquina acessa estes dados e os interpreta para descobrir quais instruções executar e quais dados manipular para a realização da tarefa. Este modelo é conhecido como Arquitetura de von Neumann[3], e é a forma mais comum dos computadores modernos.



A implementação de um computador de programa armazenado necessita de uma parte física (*hardware*) que realiza a tarefa, e de uma parte lógica (*software*) que define como realizá-la.

3 Hardware

O hardware é um conjunto de componentes e equipamentos que compõem um sistema computacional, que seguem a arquitetura von Neumann. Os componentes físicos têm diferentes propósitos.

Unidade Central de Processamento é o dispositivo que interpreta e executa e instruções, lê e escreve a memória. Seus componentes básicos são:

UC: determina quais operações serão realizadas e em que ordem.

ULA: realiza operações aritméticas e lógicas

Os registradores, que armazenam os dados sendo processados no momento, também podem ser considerado parte da UCP, mas são conceitualmente parte da memória. O desempenho da UCP era ligado a sua velocidade, mas certos limites já foram alcançados e, atualmente, os esforços são direcionados a outras formas de melhoria.

Memória é o dispositivo que permite armazenar dados em um conjunto ordenado de *bits*. A armazenagem pode ser *volátil* (perdem seus dados com ausência de energia) ou *não voláteis*. Além disso, há certa hierarquia de tipos, pois o custo é inversamente proporcional a velocidade de acesso.

A chamada *memória principal* contém os dados sendo manipulados na tarefa em execução (na memória volátil), portanto é mais rápida e intermedia o acesso aos dados da *memória secundária*. Esta serve para armazenamento permanente de dados, tem maior capacidade mas desempenho inferior.

A característica mais interessante da memória principal é o *acesso aleatório*, em que posições específicas de memória são acessadas diretamente, utilizando uma lógica de endereçamento. Isto, e a velocidade do acesso, contrastam com a memória secundária.

Entrada/Saída os dispositivos de entrada e saída de dados (E/S ou I/O) permitem a comunicação com o computador, de modo que ele possa receber dados para realizar uma tarefa, e fornecer os resultados desta. Existem diversas formas de transmitir os dados:

- Entrada: teclado, mouse, microfone, scanner, leitor de código de barras, câmera, joystick, etc.;
- Saída: monitor, caixas de som, impressora, etc.;
- E/S: disco rígido, monitor sensível a toques, pendrive, etc..

Barramento é o sistema de comunicação que transfere dados entre os componentes do computador. A taxa de transferência do barramento é o que define seu desempenho.

Fonte de Alimentação que regula a distribuição de energia para os componentes.

Placa Mãe conecta os componentes do computador de modo que possam funcionar em conjunto, possibilitando a comunicação entre eles e a distribuição de energia.

MainFrame/Supercomputador são computadores grande porte, geralmente dedicados a aplicações específicas que demandam o processamento de grandes volumes de informação.

A "nuvem" é o uso de computadores e servidores compartilhados e interligados por meio da Internet, seguindo o princípio da computação em grade.

4 Software

Software é a parte lógica do sistema computacional, que define quais instruções devem ser executadas pelo hardware. Pode ser dividido em dois tipos, *programas de sistema* que possibilitam a interação com o hardware (computador e periféricos), e *aplicações* que realizam tarefas mais específicas como edição de documentos, organização de dados, entretenimento, educacional, manipulação de mídia, e outros.

Programas de Sistema O sistema operacional é o conjunto de programas que gerencia o hardware e fornece serviços comuns às aplicações. Podem ser de tempo-real, multi-usuário, multi-tarefas, embarcado, entre outros.

Os sistemas mais conhecidos são: Unix (BSD, GNU-Linux, OS X), Microsoft Windows, Plan 9, Android, etc.

É possível ter uma ideia da utilização de cada sistema por sua fatia de mercado. Atualmente, o Windows domina o mercado de desktops, e o de dispositivos móveis é dividido entre Android e iOS. Mas a computação de alto desempenho é praticamente toda em Unix.

Utilitários São os programas que auxiliam a analisar e configurar o computador, como análise/manutenção de disco, análise de memória, anti-vírus, armazenamento de dados, cifragem, compressão, conectividade com a rede, cópia de segurança, gerenciador de arquivos, monitoração do processador, sincronização de dados/arquivos, entre muitos outros.

Controladores de Dispositivos Também conhecidos como *drivers* de dispositivos, são os programas que gerenciam um hardware específico do computador, como barramento, impressora, máquinas digitais, placa de rede/áudio/vídeo/aquisição de sinais, scanner, smartphones, etc.

Interface com Usuário São os programas que interagem com o usuário, geralmente divididos em função do uso via linha de comando ou interface gráfica.

Desenvolvimento São os programas que auxiliam o desenvolvimento de programas, como compiladores, interpretadores, IDEs, montadores, depuradorees, etc.

Aplicações São os programas com fins específicos, como navegadores, programa para escritório, entretenimento, e muitos, muitos mais.

5 Programação

"*Programação de computadores é muito divertida. Como música, é uma habilidade originada em um talento desconhecido e prática constante.*"

Larry O'Brien & Bruce Eckel

A programação é o processo de transformar o algoritmo em instruções em uma linguagem de programação, um conjunto de termos (vocabulário) e regras (sintaxe) que permite a formulação de instruções a um computador.

> *É mais importante compreender os fundamentos e técnicas da programação que dominar uma linguagem específica.*

Toda linguagem de programação é baseada em instruções *primitivas*, as entidades mais simples que ela trata; em *formas de combinação* destes elementos simples, e em *formas de abstração* destas combinações de modo que possam também ser consideradas como primitivas [4]. Por exemplo, pode-se combinar muitas instruções primitiva necessárias para exibir uma mensagem na saída padrão de tal forma que este processo seja fácil e intuitivamente obtida por uma nova primitiva: `printf`.

Alan Turing mostrou que 6 primitivas são suficientes para computar qualquer coisa [computável]. Assim, uma linguagem de programação [*Turing-completa*] define tanto a sintaxe quanto a semântica necessárias para traduzir o pensamento computacional em passos que um computador pode executar.

As linguagens modernas têm um conjunto maior e mais conveniente de primitivas que as 6 de Turing, e permitem que você as agrupe para elaborar programas mais complexos. E estas novas abstrações também podem ser agrupadas, e assim sucessivamente. Por exemplo, as primitivas mais comuns são: números (1, -2, 3.14), símbolos ('@', "Alan M. Turing", ":)"), e operações simples ($+$, $-$, $\times$, $\div$).

A sintaxe da linguagem indica quais expressões são simbolicamente corretas (`1 + 1`), ou não (`João÷`). A semântica lida com o significado associado a uma expressão sintaticamente correta, que pode ser logicamente correto (`dois = 1 + 1`) ou não (`area_do_quadrado = lado + lado`). Erros sintáticos são relativamente fáceis de se encontrar, mas o mesmo n]ao pode ser dito sobre erros semânticos.

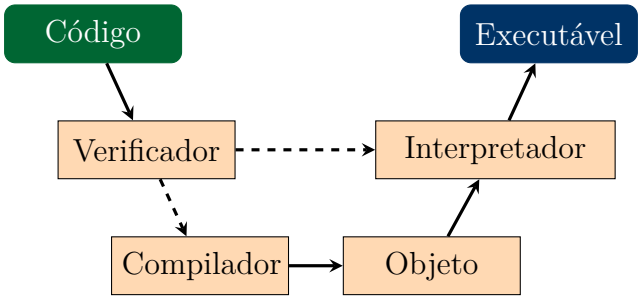
Linguagem de Máquina é a linguagem que o hardware [específico] entende (código binário). A vantagem é que não é preciso gastar esforços traduzindo instruções, mas é praticamente incompreensível para humanos [normais].

Linguagem de Baixo Nível usa instruções mnemônicas para tentar facilitar a programação. A vantagem é ser muito mais inteligível que a de máquina, mas também é pouco amigável e, dependendo do hardware utilizado, precisa ser traduzida para linguagem de máquina.

Linguagem de Alto Nível usa um vocabulário mais rico para facilitar programação. É "facilmente" aprendida e independente do hardware, mas precisa ser traduzida para linguagem de máquina.

As linguagens de alto nível claramente possibilitam programas mais breves e legíveis, mas o fazem pela composição e abstração das primitivas da máquina. Conhecendo uma linguagem de programação, pode-se elaborar programas bem interessantes. O exemplo tradicional é o "Hello World!".

De posse deste um programa sintaticamente correto, é preciso traduzí-lo da linguagem de alto nível para a linguagem de máquina para que haja instruções que o computador usado consiga interpretar (nem todas as máquinas falam a mesma linguagem).



Neste processo, o código fonte é *verificado* para garantir que não contém erros sintáticos. Dependendo da implementação da linguagem, este código é *compilado*, gerando um *objeto* (a versão das instruções originais em binário) que, por sua vez, é interpretado por um programa gerando um arquivo que pode ser executado.

Qualquer linguagem de programação pode ser interpretada ou compilada, estas são formas de *implementação* (e não características da linguagem) que não são, necessariamente, exclusivas. A *interpretação* é realizada por um programa específico (o interpretador) que realiza as instruções pelo programa sendo interpretado de modo que este seja executado. Imagine que, a cada passo, uma instrução é interpretada e executada. Já a *compilação* (realizada pelo programa compilador) traduz completamente o programa original em código objeto que pode ser executado pela máquina.

A interpretação tende a ser mais portátil, e tem a "vantagem" de realizar um passo de cada vez, já a compilação pode a gerar código otimizado e que não precisa ser interpretado a cada instrução. Outra diferença é que interpretadores são mais simples de se programa que compiladores.

Existem diversas ferramentas para realizar este processo, mas esta disciplina tem foco na linguagem `C`com o compilador GCC. No caso do exemplo, supondo que se esteja no diretório do código fonte, bastaria executar na linha de comando:

```
$ gcc hello_world.c -o hello_world
$ ./hello_world
```

6 Histórico

A história da Ciência da Computação é antiga (e muito interessante), e está entrelaçada com as histórias dos números, do hardware, de algoritmos e lógica, e da programação.

Referências

[1] SBC. Currículo de referência da sbc para cursos de graduação em bacharelado em ciência da computação e engenharia de computação. `http://www.sbc.org.br/documentos-da-sbc/category/131-curriculos-de-referencia`. Acessado em: 2015-02-14.

[2] Thomas H. Cormen. *Algorithms unlocked*. The MIT Press, Cambridge, Massachusetts, 2013.

[3] M.D. Godfrey and D.F. Hendry. The computer as von Neumann planned it. *IEEE Annals of the History of Computing*, 15(1):11–21, 1993.

[4] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and interpretation of computer programs*. MIT Press ; McGraw-Hill, Cambridge, Mass.; New York, 1996.