Introdução a programação em C.

Número ISBN: 978-85-917609-0-9

Agradecemos a Deus, aos mestres que fizeram parte de nossa formação ao nossos pais, especialmente ao pai e amigo Pedro Marconi que não está mais entre nós, aos nossos amigos as nossas companheiras e a todos que colaboraram diretamente e indiretamente para conclusão desta obra .

Coleção “ Programação C ”.

Guilherme Marconi

Ricardo Akira Harada

Prefixo Editorial: 917609





ISBN do Anais do ENEAC 2013: 978-85-7426-129-4

Sumário

1 Introdução 6
1.1 Organização Básica de um Computador 6
1.1.2 Sistemas Operacionais 7
1.1.3 Linguagens de Programação 8
1.2 Ambiente de Desenvolvimento Integrado Dev-C++ 10
1.3 Lista de Compiladores C 21
1.4 Conceitos Básicos de Programação 22
1.4.1 Estrutura Geral de um Programa 23
1.4.2 Exemplo de Programa 1 25
1.4.3 Exemplo de Programa 2 31
1.4.4 Informaç 33
1.4.5 Mapeamento de Algoritmo para Programa C 36
1.4.6 Observações Finais sobre a Linguagem C 38
2 Bases Numéricas, Expressões e Comandos Condicionais 39
2.1 Mudança de Base 40
2.1.1 Representação na Linguagem C 42
2.2 Expressões 43
2.2.1 Expressões e Operadores Aritméticos 43
2.2.2 Expressões e Operadores Relacionais 48
2.2.3 Expressões e Operadores Lógicos 52
2.2.4 Explicações Adicionais 55
2.2.5 Mapeamento de Algoritmo para Programa C 60
2.3 Comandos Condicionais 64
2.3.1 Comando Condicional if-else 65
2.3.2 Comando switch 70
2.3.3 Informações Relevantes 72
2.3.4 Mapeamento de Algoritmo para Programa C 75
3 – Considerações finais 80

1 Introdução

Sistemas operacionais assim como linguagens de programação, farão parte do contexto de aprendizagem durante a leitura da próxima seção que abrangerá uma introdução da organização básica de um PC. Para este volume, você irá aprender a configurar e instalar a plataforma que iremos desenvolver, usaremos o Dec-C++, já que em qualquer versão, não implica nos conceitos de programação em si, a leitura está organizada para entendermos como é o ambiente de programação, a estrutura geral e mapeamento da linguagem C.

Esta Obra faz parte de uma coleção de 3 livros, ao final da leitura, será indicado a você qual livro o mais indicado para continuar seus estudos na Linguagem C.

1.1 Organização Básica de um Computador

Atualmente, você encontrará em uma loja de informática diversas configurações e marcas de computador. Você possivelmente encontrará desde computadores mais simples e baratos, por exemplo computadores voltados ao processamento de dados em um escritório, até computadores mais sofisticados e caros, por exemplo computadores projetados especificamente para o mercado de jogos e processamento gráfico. Em algumas lojas, você também encontrará configurações de alto desempenho, tais como servidores de banco de dados. O entendimento do funcionamento completo de um computador disponível em uma loja de informática é muito complexo e está fora do escopo desta disciplina. Ao invés disso, você aprenderá sobre o funcionamento de um computador simplificado, cujo princípio de funcionamento é similar ao funcionamento de computadores reais. Formalmente, nós chamaremos este computador simplificado de **modelo de computador**. A Figura 1 ilustra o modelo de computador que usaremos nesta disciplina, o qual possui os seguintes componentes: processador, memória primária (também conhecida como memória principal, RAM – *random access memory* ou simplesmente memória), dispositivos de entrada/saída de dados e a interface de entrada /saída de dados.

O processador de um computador é responsável pela transformação de **dados de entrada** em **dados de saída**. Portanto, este componente possui capacidade de processamento de dados. O processador é considerado o cérebro do computador e permite a computação de, por exemplo, 10 + 20 = 30. Um processador pode realizar milhões e até bilhões de operações por segundo.
Para realizar qualquer processamento ou cálculo, o processador precisa ler dados de entrada (por exemplo, os valores 10 e 20) e escrever dados de saída (por exemplo, a soma 30). Estes dados de entrada e de saída ficam armazenados dentro do computador na memória primária. Esta memória possui quatro características básicas. A primeira característica refere-se à sua capacidade de armazenamento. A memória primária de um computador desktop (i.e., de um computador de mesa usado em ambientes de escritório e doméstico) possui uma capacidade de armazenamento entre 256 MB (i.e., milhões de bytes) e 4 GB (i.e., bilhões de bytes).

Um byte corresponde a menor unidade de armazenamento na memória. Um caractere ocupa tipicamente 1 byte na memória primária. A segunda característica refere-se à sua volatilidade. Os dados da memória primária são perdidos na ausência de energia elétrica, ou seja, os dados são perdidos depois que o computador é desligado. A terceira característica é a sua velocidade ou tempo de acesso a memória. O tempo de acesso é medido em nano segundos (ns) que corresponde a 10^{-9} segundos. O tempo de acesso da memória primária é relativamente rápido quando comparado com o tempo de acesso de outros meios de armazenamento, tais como discos magnéticos e dispositivos óticos (CD e DVD). A quarta característica da memória primária é que o tempo de acesso a qualquer posição da memória é o mesmo, ou seja, é constante. Esta última característica simplifica o planejamento do acesso aos dados, pois pode-se armazenar em qualquer posição sem que isto afete o desempenho na recuperação dos dados. Para você conseguir usar um computador e depois visualizar os resultados produzidos, é necessário utilizar dispositivos de entrada/saída de dados. Os dados são tipicamente passados para um computador pelo teclado, o qual é considerado o dispositivo de entrada de dados padrão. No entanto, os dados de entrada também podem ser obtidos a partir, por exemplo, do mouse ou de um scanner. Já os dados produzidos por um computador são comumente visualizados em um monitor de vídeo, o qual é considerado o dispositivo de saída de dados padrão. Outro dispositivo de saída de dados bastante utilizado é a impressora. Em particular, discos magnéticos podem atuar tanto como dispositivos de entrada de dados quanto como dispositivos de saída de dados. Discos magnéticos (também conhecidos como hard disks ou winchesters) são usados para armazenamento persistente de dados (i.e., dados não voláteis) e são classificados como memória secundária. Por fim, a interface de entrada/saída de dados possibilita a comunicação entre os dispositivos de entrada/saída de dados e o processador. Na Figura 1, esta interface é representada pela seta dos dispositivos de entrada de dados para o processador e pela seta do processador para os dispositivos de saída de dados.

1.1.2 Sistemas Operacionais

O gerenciamento dos **recursos computacionais** (por exemplo, processador, memória primária e discos magnéticos) é realizado por um programa especial chamado **sistema operacional**. O sistema operacional é o primeiro programa a ser executado após o computador ser ligado. Ele literalmente assume o controle do computador para permitir que os recursos computacionais possam ser usados corretamente e compartilhados por diversos outros programas, tais como programas aplicativos e utilitários, ambientes de desenvolvimento integrado, compiladores, interpretadores e jogos.

O uso do sistema operacional para controlar o computador apresenta várias vantagens. A primeira vantagem refere-se ao correto funcionamento do computador. Sistemas operacionais são desenvolvidos por equipes especializadas, garantindo que o computador funcione sem erros e de forma segura. Possíveis erros (também chamados de bugs do sistema), assim que identificados são corrigidos pelo fabricante do sistema operacional e a correção é

disponibilizada para os usuários por meio de pacotes de atualização (i.e., comumente via Web, tal como feito no Windows Update).

A segunda vantagem do uso de um sistema operacional consiste na rapidez do gerenciamento dos recursos, desde que os componentes de um sistema operacional são customizados para este fim.

Uma terceira vantagem do uso de um sistema operacional é que o computador torna-se mais fácil de usar pelos usuários finais e também pelos programadores. Por exemplo: acessar e interagir diretamente com um disco *magnético é uma atividade tediosa, sujeita a erros e com muitos detalhes dependentes do dispositivo* físico (i.e., troca-se o modelo de disco magnético, alteram-se estes detalhes). Ao invés disso, o sistema operacional oferece serviços para ler e escrever em um disco magnético. O uso desses serviços pelos programadores é muito mais fácil.

Os detalhes da comunicação com o disco magnético são resolvidos de forma transparente pelo sistema operacional. Outro exemplo é o uso de um computador por um usuário leigo, ou seja, um usuário que possui pouca familiaridade com computadores. Para um usuário leigo, é muito mais fácil interagir com o computador por meio de uma interface gráfica constituída por janelas, botões e ícones, do que interagir por meio de uma linha de comandos (i.e., *prompt* para digitação de comandos em um interpretador de comandos ou *shell*). Nenhuma interface gráfica é fornecida pelo computador.

Cada sistema operacional disponibiliza a sua própria interface gráfica (ou diversas opções de interface gráfica) para interação com os usuários. Existem diversos sistemas operacionais no mercado. Na plataforma de computadores PC (*Personal Computer*), os sistemas operacionais mais usados pelos usuários são os sistemas da família Windows da Microsoft (Windows 98, Windows 98SE, Windows XP, Windows NT e Windows Vista), da família Linux (distribuições *SUSE* da *Novell, red hat e Fedora da Red Hat, Inc., Ubuntu da Canonical Ltd., CentOS da The Community ENTerprise Operating System e Debian da Public Interest, Inc.) e da família Mac OS da Apple* (atualmente na versão Mac OS X 10.4.1). Há também sistemas operacionais específicos para outras plataformas de computadores, por exemplo z/OS para os servidores System z da IBM. Nesta disciplina usaremos computadores da plataforma PC com um sistema operacional da família Windows, preferencialmente o sistema operacional Windows XP ou Windows 7.

1.1.3 Linguagens de Programação

Um computador processa instruções escritas apenas em linguagem de máquina. Uma instrução escrita em linguagem de máquina é composta por uma sequência de zeros e uns (i.e., sequência de bits) que tem significado para o computador. Por exemplo, a instrução formada por 00001100 11001100 00110011 pode significar em um computador hipotético SOME 10 e 20 (obs.: note no exemplo o agrupamento em conjuntos de 8 bits, que formam um byte).

É fácil perceber que desenvolver uma solução para um problema complexo usando linguagem de máquina não é simples tampouco intuitivo para nós seres humanos. Ao invés de

usar diretamente a linguagem de máquina, programadores usam linguagens de mais alto nível desenvolvidas por fabricantes de software ou disponibilizadas gratuitamente por organizações de padronização ou de *open-source*. Estas linguagens de mais alto nível possuem construtores que tornam mais direta a concepção do raciocínio lógico da solução. Exemplos de linguagens de mais alto nível são: C (linguagem que será usada nesta disciplina), C++ (extensão da linguagem C que provê suporte ao paradigma de orientação a objetos e não será tratada nesta disciplina), Java, COBOL, Pascal, Delphi, Perl, C# e PHP.

Neste momento, você provavelmente estará se perguntando sobre a seguinte dúvida. Se o computador não entende outra linguagem a não ser a linguagem de máquina, como o computador irá processar instruções escritas em uma linguagem de mais alto nível ? Como o computador irá entender instruções escritas na linguagem C? A resposta é: o computador não irá entender e não irá processar diretamente instruções escritas em uma linguagem de mais alto nível, tal como na linguagem C. Para que isto seja possível, é usado um **processo de tradução** da linguagem de mais alto nível para a linguagem de máquina. Assim, instruções na linguagem C são traduzidas em um conjunto de instruções equivalentes em linguagem de máquina, antes de ser processadas pelo computador. O processo de tradução pode ocorrer de duas formas básicas. A primeira forma chama-se **interpretação**. Na interpretação, cada instrução do código-fonte é interpretada e traduzida por um programa chamado **interpretador** em um conjunto de instruções equivalentes em linguagem de máquina. Logo em seguida, o interpretador executa este conjunto de instruções e produz algum resultado. O uso da interpretação apresenta como principal vantagem o fato de que a escrita de programas torna-se mais interativa.

Por exemplo, não é necessário escrever todo o programa para obter resultados parciais. Além disso, no início da aprendizagem de uma linguagem de programação, você provavelmente cometerá muitos erros na escrita das instruções. Na interpretação, os erros são identificados à medida que as instruções são escritas, novamente sem a necessidade de se ter o programa completo. Em teoria, isto ajuda no aprendizado em estágios iniciais. Na prática, porém, o uso de interpretadores apresenta sérias desvantagens, tais como visibilidade do código-fonte (i.e., não protege-se que outras pessoas vejam o que está escrito no programa), perda de desempenho (i.e., o código gerado e executado é muito mais lento quando comparado ao processo de compilação.

Por exemplo, se uma mesma instrução é executada 500 vezes, o interpretador tem que interpretá-la e depois executá-la 500 vezes) e necessidade de se ter o programa interpretador em cada computador que for executar o programa. Um bom interpretador para a linguagem C, chamado "*Embedded Ch*", está disponível para download gratuito em: http://www.softintegration.com/.

Outra forma de se realizar o processo de tradução chama se **compilação**. Na compilação, o código-fonte deve estar totalmente escrito e sem erros. Um programa chamado **compilador** traduz todas as instruções do código-fonte, gerando o código objeto. Este código é chamado de programa-objeto, possui extensão .OBJ e apesar de já estar em linguagem de máquina ainda não pode ser executado pelo computador. Em seguida, um outro programa, chamado **link-editor**, adiciona bibliotecas ao código-objeto que permitem que este código possa ser carregado em memória primária e depois executado. O código resultante é chamado de programa executável e

possui extensão .EXE na plataforma Windows. De forma contrária ao processo de interpretação, o processo de compilação esconde o código-fonte, produz um programa mais rápido e não necessita de nenhum programa adicional para executar o programa gerado no processo de compilação. Apesar do processo de compilação envolver alguns passos e programas diferentes (e.g. compilador e link-editor), em geral estes passos e programas estão contidos em um único ambiente, chamado ambiente de desenvolvimento integrado (IDE – *Integrated Development Environ ment*). Um IDE possui, além de um compilador e um link editor, um editor de texto customizado para a escrita do código fonte (por exemplo, o editor pode usar uma cor específica para representar as palavras reservadas da linguagem), ferramentas de depuração de erros, ferramentas de ajuda, dentre outras. Nesta disciplina, nós utilizaremos o IDE Bloodshed Dev-C++ 5. Na próxima seção, você aprenderá mais sobre o Dev-C++. Por fim, entende-se por **código-fonte** um programa escrito em uma linguagem de mais alto nível usando um editor de texto. Este programa (chamado **programa-fonte**) é comumente armazenado em um arquivo texto em um disco magnético ou em outras mídias de armazenamento persistente (tais como floppy-disk, CD, DVD e pen-drive). O código-fonte de um programa C possui por convenção a extensão .C (como exemplo, "programa1.c").

1.2 Ambiente de Desenvolvimento Integrado Dev-C++

Para nosso estudo, usaremos o ambiente de desenvolvimento integrado Dev-C++ versão, esta seção descreve como **instalar** o Dev-C++ em um computador. Além disso, esta seção ensina como usar o Dev-C++ para **digitar** um programa-fonte na linguagem C e depois **compilar** e **executar** o programa. Relembrando que a **compilação** de um programa tem por objetivo **traduzir** um programa-fonte em um programa executável. Um programa-fonte é mais fácil de ser codificado e compreendido por nós humanos, desde que este programa é escrito em uma linguagem de programação de mais alto nível (por exemplo, escrito na linguagem C), a qual utiliza termos já conhecidos em nossa linguagem (em geral os termos são da língua inglesa, como exemplo os termos *begin, end, for e while*). Porém, um programa-fonte não pode ser diretamente executado por um computador. De forma contrária, um programa executável possui código binário que pode ser carregado e processado diretamente por um computador.

Porém, o uso de código binário na escrita de programas dificulta a elaboração e a organização da lógica que resolverá um certo problema. Desta forma, programadores escrevem código em um programa-fonte que depois será traduzido por um **compilador** em um programa executável.

Um ambiente de desenvolvimento integrado, tal como Dev-C++ 5, possui um compilador e outras ferramentas úteis para a escrita de um programa (e.g., editor de texto adaptado para a linguagem de programação, ferramenta de depuração de programas que auxilia na identificação e remoção de erros de programação, um conjunto de bibliotecas com código que pode ser reaproveitado e a documentação da linguagem). A **compilação** de um programa somente acontece se o programa- fonte estiver sem erros. Os erros podem ser léxicos, sintáticos ou