



Centro Universitário Cidade Verde

Estrutura de dados



AUTORIA

Ricardo Bittencourt Socreppa

Bem vindo(a)!

O Objetivo principal deste material é capacitá-lo a compreender os princípios básicos da lógica de programação (desenvolver o raciocínio lógico aplicado à solução de problemas em nível informatizado), bem como os principais comandos necessários ao desenvolvimento de algoritmos em uma linguagem de programação, para que o mesmo possa estruturar e desenvolver a resolução de problemas computacionais.

Além de formar um profissional ético, competente e comprometido com a sociedade em que vive, ou seja, com o desenvolvimento de perspectivas críticas, integradoras, e que possa construir sínteses contextualizadas.

Também vamos conhecer e usar os fundamentos, os métodos e as técnicas da lógica de programação. Compreender e empregar estruturas de dados complexas. Ampliar o domínio de conhecimento algoritmos. Elaborar códigos otimizados. Avaliar a utilização de estruturas de dados no contexto empresarial.

Preparados para esta Jornada?

Venham comigo e tenham todos uma ótima leitura!



| Unidade 1

Introdução a Programação e Linguagem C



AUTORIA

Ricardo Bittencourt Socreppa

Introdução

Prezados alunos(as)!

Veremos neste capítulo uma introdução a algoritmos e programação, e serão abordados os princípios que embasam esse conteúdo. Para acompanhar não é necessário ter um conhecimento prévio de programação ou ser um hacker de computadores. Parte-se do preceito que o leitor sabe somente como interagir com o computador, ligar, desligar, instalar e utilizar programas e acessar a internet. Sendo assim, pretende-se com este capítulo estimular a curiosidade por esta disciplina, introduzindo os conceitos básicos, de forma que o leitor entenda a importância da programação e da linguagem de programação em C.

Da mesma forma que existem vários idiomas em todo o mundo, na computação, existem várias linguagens de programação. A linguagem de programação é utilizada para escrever programas e a linguagem faz uma tradução do algoritmo para uma linguagem que o computador entenda.

Neste texto, você vai estudar sobre a linguagem C, por ser bastante difundida no meio acadêmico e usada para que você dê os primeiros passos no mundo da programação.

Você vai entender como surgiu a linguagem por meio do histórico da linguagem C, compreender a estrutura básica da linguagem, além de conhecer o ambiente de desenvolvimento Falcon C++ e configurá-lo.

Venha comigo!



Introdução a Programação



AUTORIA

Ricardo Bittencourt Socreppa

Este tópico tem como finalidade apresentar alguns conceitos básicos, como o hardware e software que formam os computadores, conceitos básicos de programação, e como são organizadas as linguagens de programação.

Computadores têm uma grande importância no mundo atual, e estão integrados a tudo, incluindo hardware e software. Os hardwares são os componentes físicos, como placa-mãe, monitor, teclado e mouse. Já os softwares consistem no sistema operacional e nos programas que rodam nesse sistema, como o iTunes, Office e Firefox. O hardware e software se complementam e formam o que chamamos de computador.

Hardware e software

O hardware consiste em três componentes principais: a unidade central de processamento (ALU, que é conhecido normalmente pela sigla em inglês CPU), a memória e os dispositivos. A CPU fica responsável por realizar as instruções dos softwares realizando cálculos aritméticos, lógicos, de controle ou operações de entrada e saída, de acordo com a instrução que está sendo processada. A memória é onde os dados são guardados, sempre usando a unidade básica de bits. Ela pode ser do tipo RAM, que são as memórias voláteis, isto é, necessitam de energia para manter a informação armazenada; ou também pode ser do tipo ROM/Flash (ou outros), que guardam os dados a todo momento, sem necessidade de retroalimentação, como o hard-drive e os pen drives.

No topo do hardware roda um sistema operacional, como o Microsoft Windows, Mac OS ou Linux. Esses sistemas conectam os softwares com o hardware, criando uma interface básica para os softwares ou aplicativos realizarem processamento e controle no hardware. Dessa forma, os sistemas operacionais devem ter os recursos de hardware mapeados e ter instruções de como usar esses recursos, o que normalmente é feito por drivers que dizem ao sistema como utilizar o hardware.

Os softwares rodam em cima do sistema operacional, e devem saber como ele opera, por isso os programas normalmente servem para um sistema operacional, e para manter a multiplataforma (isto é, rodar em diferentes plataformas) eles devem endereçar aspectos específicos de cada sistema operacional.

Um software é escrito a partir de um código-fonte, que contém as instruções de acordo com uma linguagem de programação. Esse código é lido e processado, de forma a se tornar as ações do computador. A criação desse código-fonte é realizada a partir da programação.



Centro Universitário Cidade Verde

Conceitos básicos de programação



AUTORIA

Ricardo Bittencourt Socreppa

A programação é o desenvolvimento de software por instruções de comando que o hardware deve realizar. As linguagens de programação transformam essas instruções de hardware em uma linguagem mais simples para os programadores. Finalmente os programadores são as pessoas que sabem ler e escrever instruções em alguma linguagem de programação – e esta obra é o primeiro passo para ajudar o leitor a se tornar um programador.

Todas as linguagens usam instruções como base. Essas instruções são seguidas literalmente pelos computadores. Dessa forma, se metaforicamente mandarmos o computador pular, teríamos que definir para ele diversas variáveis, como a forma de realizar o pulo, a qual altura ele deve pular, qual o impulso ele deve ter, onde ele deve cair etc.

Como exemplo de instruções, poderíamos colocar alguns exemplos comuns: “faça isso; depois faça aquilo”; “se essa condição for verdadeira: faça isso; caso contrário faça aquilo”; “repita essa ação um número determinado de vezes”; “realize uma ação até eu mandar parar”.

Existem diversas linguagens de programação e a cada ano aparecem mais. Algumas são melhorias para desenvolvimento web, outras para desenvolvimento de aplicativos e outras para desenvolvimento científico. As linguagens podem ser divididas em três tipos gerais:

- **Linguagem de máquina:** são diretamente entendidas pelos computadores (linguagem binária em 1s e 0s), conforme o código, eles executam as instruções necessárias.
- **Linguagem Assembly:** elas abstraem os códigos de máquina em instruções que representam as operações elementares que o computador realiza.
- **Linguagens de alto nível:** têm uma linguagem mais próxima da linguagem humana, aumentando a performance que um programador leva para escrever um software. As linguagens modernas de programação são desse tipo, sendo Python uma delas.

As linguagens de alto nível podem ser compiladas, transformando a linguagem diretamente em código de máquina, um processo que pode tomar tempo, dependendo do tamanho do código a ser compilado, mas tem a melhor performance quando comparado com as outras alternativas. Essas linguagens também podem ser interpretadas, e não têm tempo de compilação, pois o código é interpretado em tempo de execução e em código de máquina, mas podem ter uma performance pior (a velocidade de execução) que as linguagens compiladas, pois perde-se tempo de processamento com a interpretação.

Com o desenvolvimento do hardware, a execução em código de máquina ou Assembly não se tornou mais viável, pois as instruções se tornaram grandes, e pela falta de claridade, ininteligíveis. Sendo assim, as linguagens de alto nível começaram com a criação das programações estruturadas, que foram criadas para ser sequências de código claras, corretas e facilmente modificadas. As mais famosas

foram o Pascal e o C, que se focam em ações, verbos, e são sequencialmente executadas, de forma a dizer o que o hardware deve realizar. Neste Capítulo utilizaremos a linguagem C, que será apresentada na sequência.

C: histórico e importância

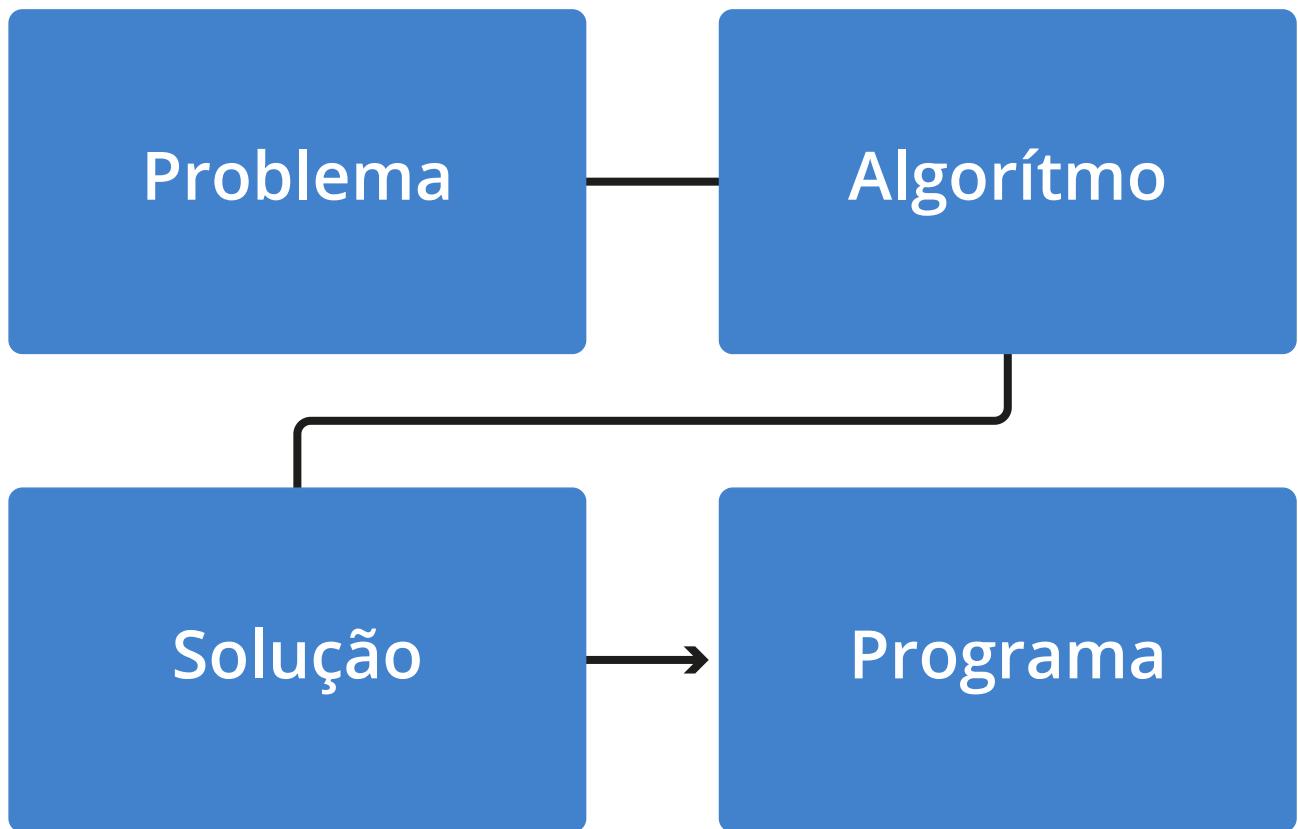
| Code | Output | Notes | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|---|---|------------------------|----------------|---------|-----|--------------|--------------|---------------------|---------|-----|--------------|--------------|----------------|--------|-----|--------------|--------------|-------------------------------|---------|-----|--------------|--------------|-----------------------|---------|-----|--------------|--------------|----------------|---------|-----|--------------|--------------|---------------------|---------|-----|--------------|--------------|----------------|--------|-----|--------------|--------------|-------------------------------|---------|-----|--------------|--------------|-----------------------|---------|-----|---|--|--|--|--|--------------|--------------|-----------|------------------------|-----------|--------------|--------------|---------|-----------------|--|--------------|--------------|--|--|--|---|
| <pre>// You can use background-image: url('path/to/image.jpg'); background: linear-gradient(radial, red 20px);</pre> | | • You can merge colors across the background, while the text runs across the top. • You can use <code>background-size</code> to change the size of the image. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <pre>// nested properties</pre> | | • You can make colors more specific. • You can build upon styles. • You can apply font-weight, font-style, font-family, color, background-color, color, width, height, and position properties. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <pre>function calculateTotal() { var sum = 0; for (var i = 0; i < data.length; i++) { sum += data[i]; } return sum; }</pre> | <table border="1"><tbody><tr><td>230300000000</td><td>000000000000</td><td>10450 Benefits</td><td>10 : 37</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>30240 Payroll taxes</td><td>10 : 32</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>70745 Salaries</td><td>8 : 36</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>70229 Commissions and bonuses</td><td>12 : 44</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>23574 Personnel total</td><td>10 : 32</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>10450 Benefits</td><td>10 : 37</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>30240 Payroll taxes</td><td>10 : 32</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>70745 Salaries</td><td>8 : 36</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>70229 Commissions and bonuses</td><td>12 : 44</td><td>NSA</td></tr><tr><td>230300000000</td><td>000000000000</td><td>23574 Personnel total</td><td>10 : 32</td><td>NSA</td></tr><tr><td colspan="5">Stock Exchange - Sys 400 Food Company (Au.), contra itemized against Mexico team</td></tr><tr><td>230300000000</td><td>000000000000</td><td>0.357697%</td><td>771 - 10.4% increasing</td><td>+453,6534</td></tr><tr><td>230300000000</td><td>000000000000</td><td>3303.29</td><td>32,755334 Items</td><td></td></tr><tr><td>230300000000</td><td>000000000000</td><td></td><td></td><td></td></tr></tbody></table> | 230300000000 | 000000000000 | 10450 Benefits | 10 : 37 | NSA | 230300000000 | 000000000000 | 30240 Payroll taxes | 10 : 32 | NSA | 230300000000 | 000000000000 | 70745 Salaries | 8 : 36 | NSA | 230300000000 | 000000000000 | 70229 Commissions and bonuses | 12 : 44 | NSA | 230300000000 | 000000000000 | 23574 Personnel total | 10 : 32 | NSA | 230300000000 | 000000000000 | 10450 Benefits | 10 : 37 | NSA | 230300000000 | 000000000000 | 30240 Payroll taxes | 10 : 32 | NSA | 230300000000 | 000000000000 | 70745 Salaries | 8 : 36 | NSA | 230300000000 | 000000000000 | 70229 Commissions and bonuses | 12 : 44 | NSA | 230300000000 | 000000000000 | 23574 Personnel total | 10 : 32 | NSA | Stock Exchange - Sys 400 Food Company (Au.), contra itemized against Mexico team | | | | | 230300000000 | 000000000000 | 0.357697% | 771 - 10.4% increasing | +453,6534 | 230300000000 | 000000000000 | 3303.29 | 32,755334 Items | | 230300000000 | 000000000000 | | | | • You can make colors more specific. • You can build upon styles. • You can make colors deeper. • You can add opacity to colors. |
| 230300000000 | 000000000000 | 10450 Benefits | 10 : 37 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 30240 Payroll taxes | 10 : 32 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 70745 Salaries | 8 : 36 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 70229 Commissions and bonuses | 12 : 44 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 23574 Personnel total | 10 : 32 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 10450 Benefits | 10 : 37 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 30240 Payroll taxes | 10 : 32 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 70745 Salaries | 8 : 36 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 70229 Commissions and bonuses | 12 : 44 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 23574 Personnel total | 10 : 32 | NSA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Stock Exchange - Sys 400 Food Company (Au.), contra itemized against Mexico team | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 0.357697% | 771 - 10.4% increasing | +453,6534 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | 3303.29 | 32,755334 Items | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 230300000000 | 000000000000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <pre>// You can + nested properties background-image: url('path/to/image.jpg');</pre> | | • You can merge colors across the background, while the text runs across the top. • You can use <code>background-size</code> to change the size of the image. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

AUTORIA
Ricardo Bittencourt Socreppa



Antes de iniciarmos a falar sobre C, que é a linguagem de programação que utilizaremos, vamos compreender o que é uma linguagem de programação. A **Figura 1**, a seguir, faz a representação de como um algoritmo pode nos auxiliar na resolução de problemas no nosso dia-a-dia.

Figura 1 - Diagrama de um algoritmo que ajuda no entendimento dos problemas.



Fonte: Adaptado de BACKES (2013).

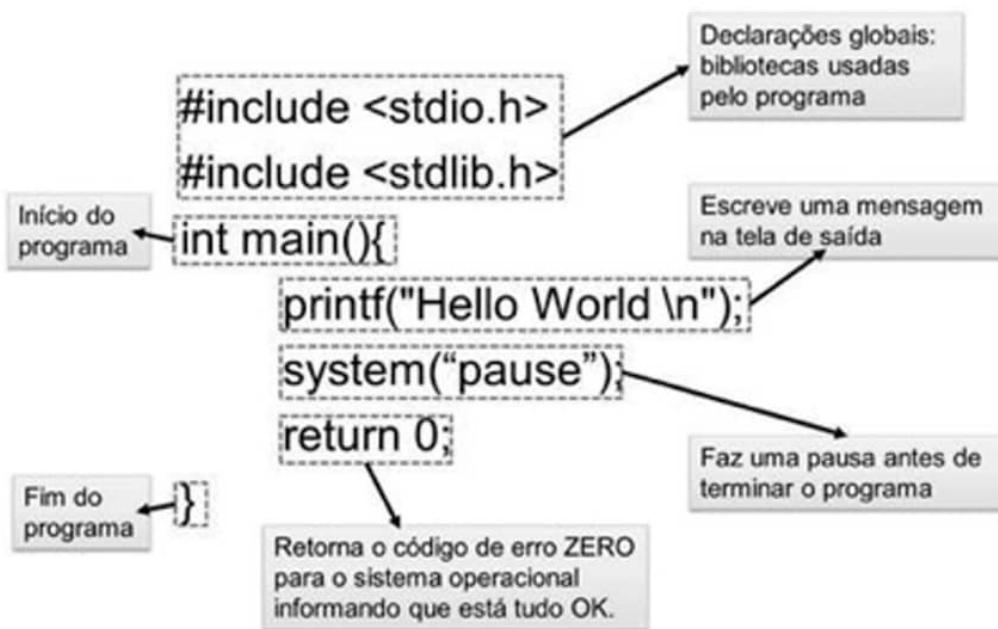
Segundo BACKES (2013), a linguagem C é uma das mais bem sucedidas linguagens de alto nível já criadas. Considera-se como linguagem de alto nível aquela que possui um nível alto de abstração, ou seja, está mais próximo da linguagem humana do que do código de máquina. Sendo considerada uma das linguagens de programação mais utilizadas de todos os tempos. Ela foi criada em 1972, nos laboratórios Bell na empresa AT&T, por Dennis Ritchie, sendo revisada pela ANSI (American National Standards Institute) em 1989. Trata-se de uma linguagem estruturalmente simples de grande portabilidade. A linguagem C é uma linguagem procedural, ou seja, ela permite que um problema complexo seja facilmente disposto em módulos, sendo cada módulo um problema mais simples. Além disso, a linguagem C permite acesso de baixo nível de memória, que permite o acesso e a programação direta do microprocessador.

Por fim, a linguagem C pode ser compilada em uma grande variedades de plataformas e sistemas operacionais com pequenas alterações no código fonte.

Esqueleto de um programa em linguagem C

Todo programa escrito em linguagem C que vier a ser desenvolvido deve possuir o esqueleto mostrado no código-fonte abaixo:

Figura 2 - Esqueleto de demonstração de um código-fonte.



Fonte: BACKES (2013).

De acordo com BACKES (2013), **no início** do programa, a região onde são feitas as suas **declarações globais**, ou seja, aquelas que são **válidas para todo o programa**. No exemplo, o comando `#include < nome_da_biblioteca >` é utilizado para declarar as bibliotecas que serão utilizadas. Todo o programa em linguagem C deve conter a **função main()**. Essa função é responsável pelo início da execução do programa, e é dentro dela que colocamos os comandos que queremos que o programa execute. **As chaves** definem o **início ("{" e o fim ("}")** de um bloco de comandos/ instruções. No exemplo, as chaves definem o início e o fim do programa. A declaração de um comando **quase sempre** termina com **ponto e vírgula ("";")**. Nas próximas seções, veremos quais comandos não terminam com ponto e vírgula. **Os parênteses** definem o **início ("(" e o fim ("")** da lista de argumentos de uma função.

Indentação do Código

Trata-se de uma **convenção de escrita de códigos-fonte** que visa a modificar a estética do programa para auxiliar a sua leitura e interpretação. O ideal é sempre criar um novo nível de indentação para um novo bloco de comandos.

Figura 3 - Exemplo de identação de um código-fonte.

```
#include <stdio.h>
#include <stdlib.h>
int main(){ printf("Hello world!\n");return 0;}
```

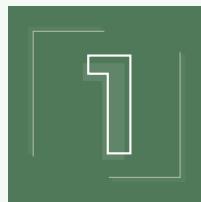
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Fonte: elaborado pelo autor.

A compilação do programa

Segundo BACKES (2013), a compilação possui 4 etapas:



Pré-processamento: antes de iniciar a compilação do nosso código-fonte, o arquivo é processado por um pré-processador. Nessa etapa, ocorrem a **remoção dos comentários** e a **interpretação das diretivas de compilação utilizadas**, as quais se **iniciam com #**.



Verificação sintática: aqui se verifica se o **código-fonte foi escrito corretamente**, de acordo com a linguagem C.



Compilação: cada arquivo de código-fonte do seu programa é processado, sendo **criado um arquivo “objeto”** para cada um deles. Nessa etapa, não é gerado nenhum arquivo que o usuário possa executar.



Link-edição: o trabalho do link-editor é **unir todos os arquivos “objeto”** que fazem parte do programa em um **único arquivo executável**, o programa propriamente dito.

Usando uma biblioteca: o comando `#include`

Como afirma BACKES (2013), o comando **#include** é utilizado para declarar as bibliotecas que serão utilizadas pelo programa. Uma **biblioteca** é um arquivo contendo um **conjunto de funções (pedaços de código), variáveis, macros etc.**, já implementados e que **podem ser utilizados pelo programador** em seu programa. De modo geral, os arquivos de bibliotecas na linguagem C são terminados com a **extensão .h**.

- O comando `#include` permite duas sintaxes:

- **#include < nome_da_biblioteca >**: o pré-processador **procurará** pela biblioteca nos **caminhos de procura pré-especificados do compilador**. Usamos essa sintaxe quando estamos incluindo uma biblioteca que é própria do sistema, como as bibliotecas stdio.h e stdlib.h.
- **#include “nome_da_biblioteca”**: o pré-processador **procurará** pela biblioteca **no mesmo diretório onde se encontra o nosso programa**.

Criando suas próprias bibliotecas

Como defende BACKES (2013), a linguagem C permite **criar nossa própria biblioteca**. Nela, podemos colocar nossas **funções, estruturas** etc., o que torna mais prática e fácil a sua **utilização em outros projetos**. Uma biblioteca é como o seu arquivo de código-fonte principal, com a diferença de que ele **não possui a função main()**. Isso ocorre porque o seu programa não vai começar na biblioteca.

- Para criar uma biblioteca em C precisamos de dois arquivos:

- **Cabeçalho (ou header) da biblioteca**: esse arquivo contém as declarações e definições do que está contido dentro da biblioteca. Sua extensão é **.h**.
- **Código-fonte da biblioteca**: arquivo que contém a implementação das funções definidas no cabeçalho. Sua extensão é **.c**.

- Criando uma biblioteca que implemente uma função para somar dois números:

Arquivo de Cabeçalho aritimetica.h

```
int soma(int a, int b);
```

Fonte: elaborado pelo autor.

Código fonte da Biblioteca aritimetica.c

```
#include "aritimetica.h"

int soma(int a, int b){
    return a + b;
}
```

Fonte: elaborado pelo autor.

Código fonte da Biblioteca aritimetica.c

```
#include "aritimetica.h"

int soma(int a, int b){
    return a + b;
}
```

Fonte: elaborado pelo autor.

Chamando a biblioteca no programa principal main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "aritimetica.h"
int main()
{
    int x,y,z;
    char ch;
    printf("Digite a operacao matematica (+): ");
    ch = getchar();
    printf("Digite primeiro numero: ");
    scanf("%d",&x);
    printf("Digite segundo numero: ");
    scanf("%d",&y);
    switch(ch){
        case '+': z = soma(x,y);
```

Fonte: elaborado pelo autor.



Preparando o Ambiente



AUTORIA

Ricardo Bittencourt Socreppa

IDEs

- Existem **diversos ambientes de desenvolvimento integrado** ou IDEs (Integrated Development Environment) que podem ser utilizados para a programação em linguagem C. Um deles é o **Falcon C++**;
- Roda somente no Windows;
- Pode ser baixado diretamente de seu site <http://falconcpp.sourceforge.net/>;

Criando um novo projeto no Falcon C++

- Para criar um novo projeto de um programa no software Falcon C++, basta seguir estes passos:

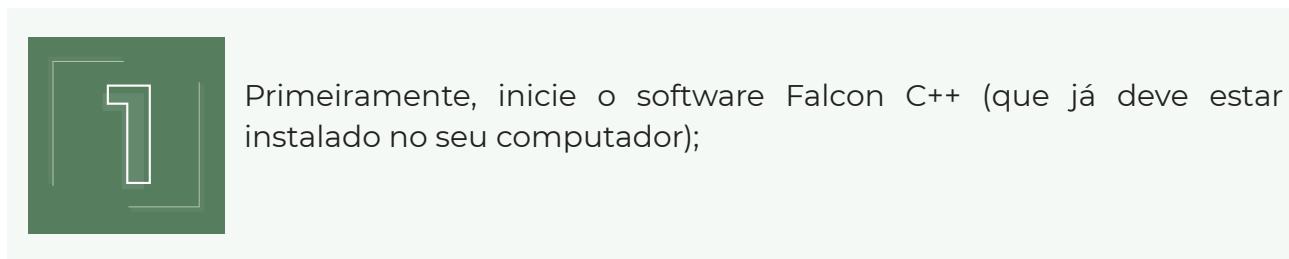
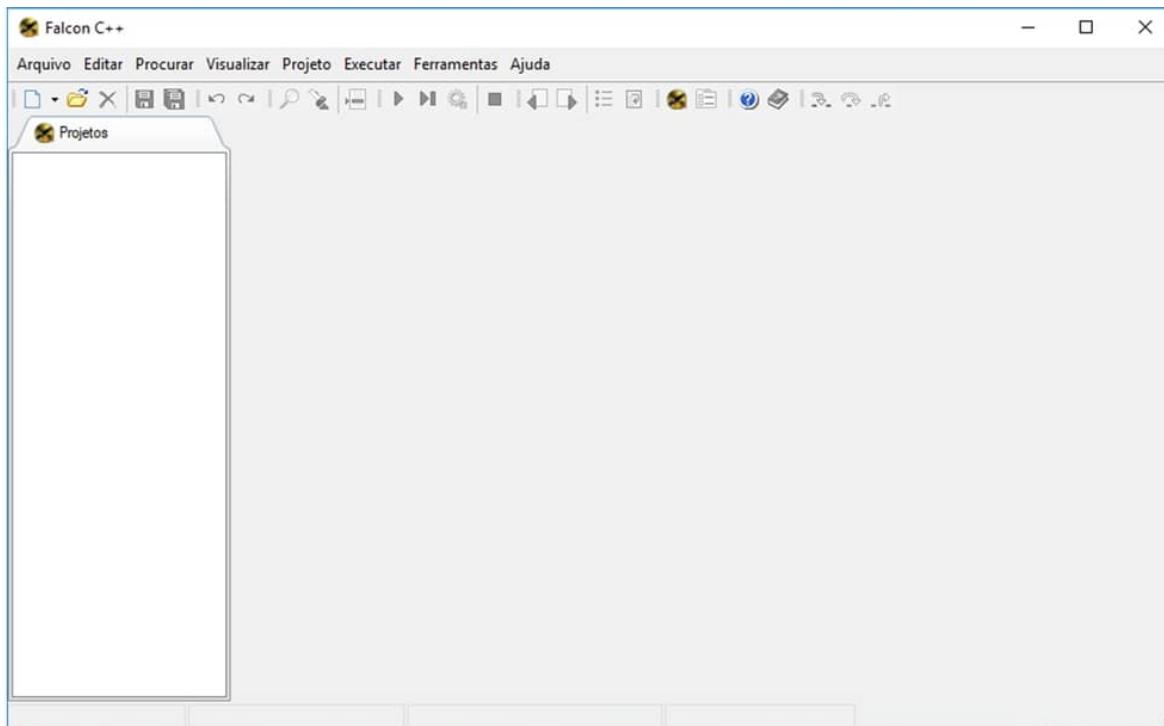


Figura 5 - Área de Trabalho do Software Falcon C++.

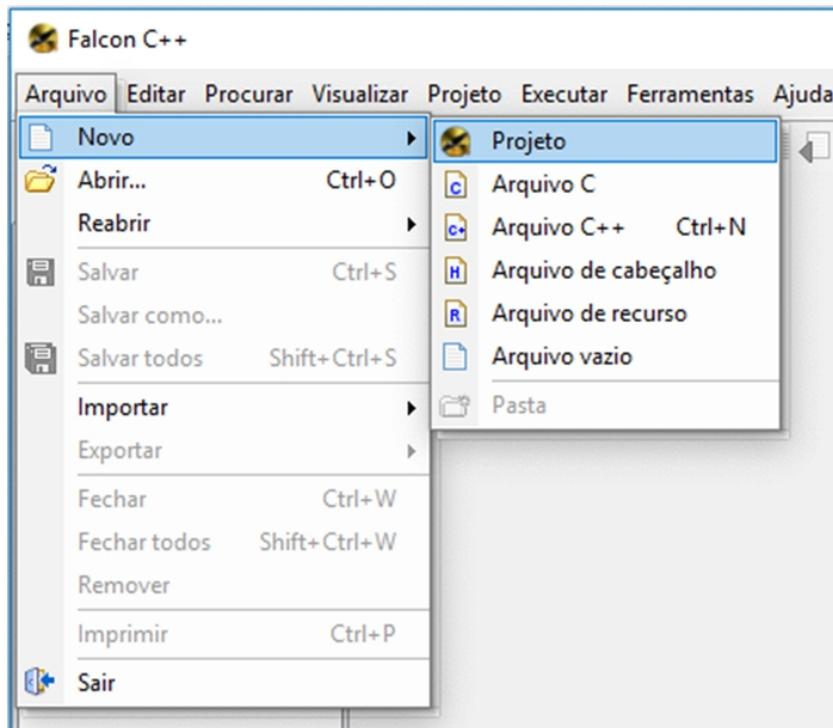


Fonte: elaborado pelo autor.



Em seguida clique em “Arquivo”, escolha “Novo” e depois “Projeto...”

Figura 6 - Criando um Novo Projeto no Software Falcon C++.

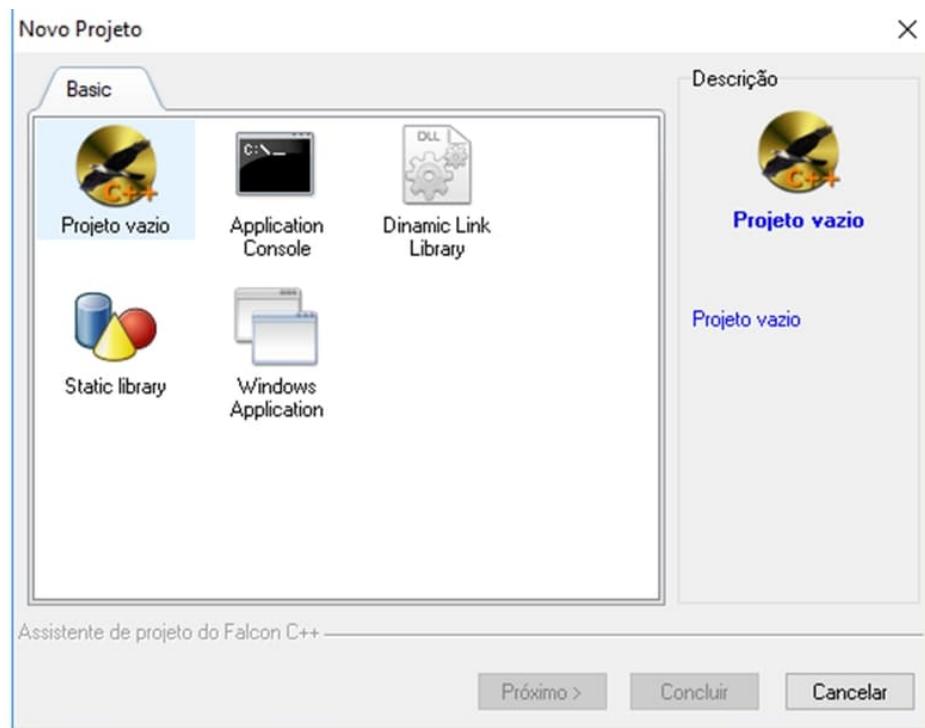


Fonte: elaborado pelo autor.



Uma lista de modelos (templates) de projetos vai aparecer. Escolha “**Aplication Console**” e no botão **próximo**;

Figura 7 - Modelos de Projeto do Software Falcon C++.

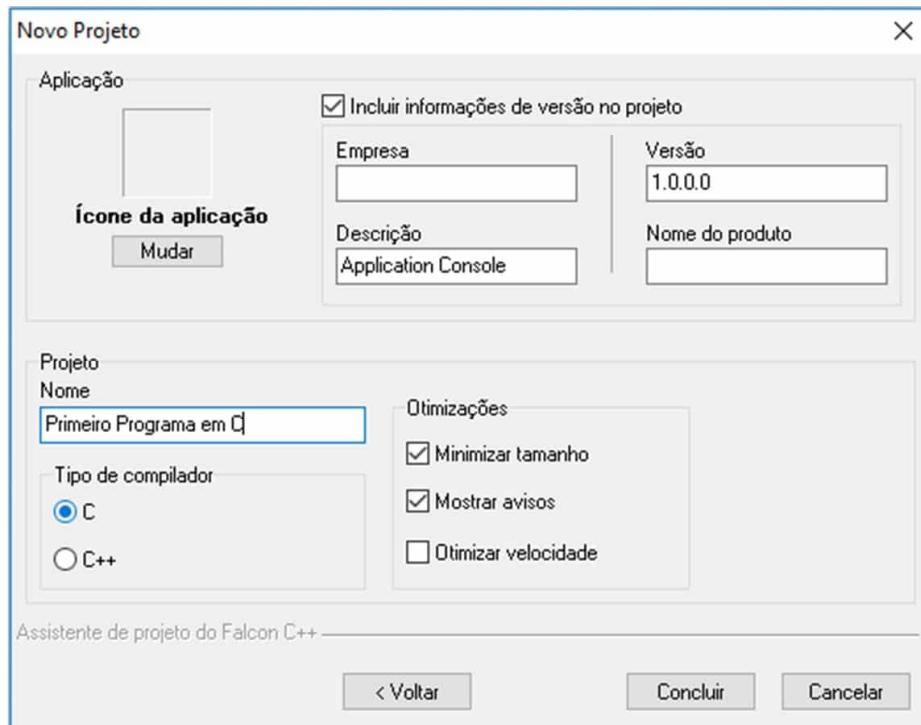


Fonte: elaborado pelo autor.



No Seção Projeto “**Nome**”, coloque um nome para o seu projeto. No tipo do compilador marque “**C**” Clique em “**Concluir**”.

Figura 8 - Modelos de Projeto do Software Falcon C++.

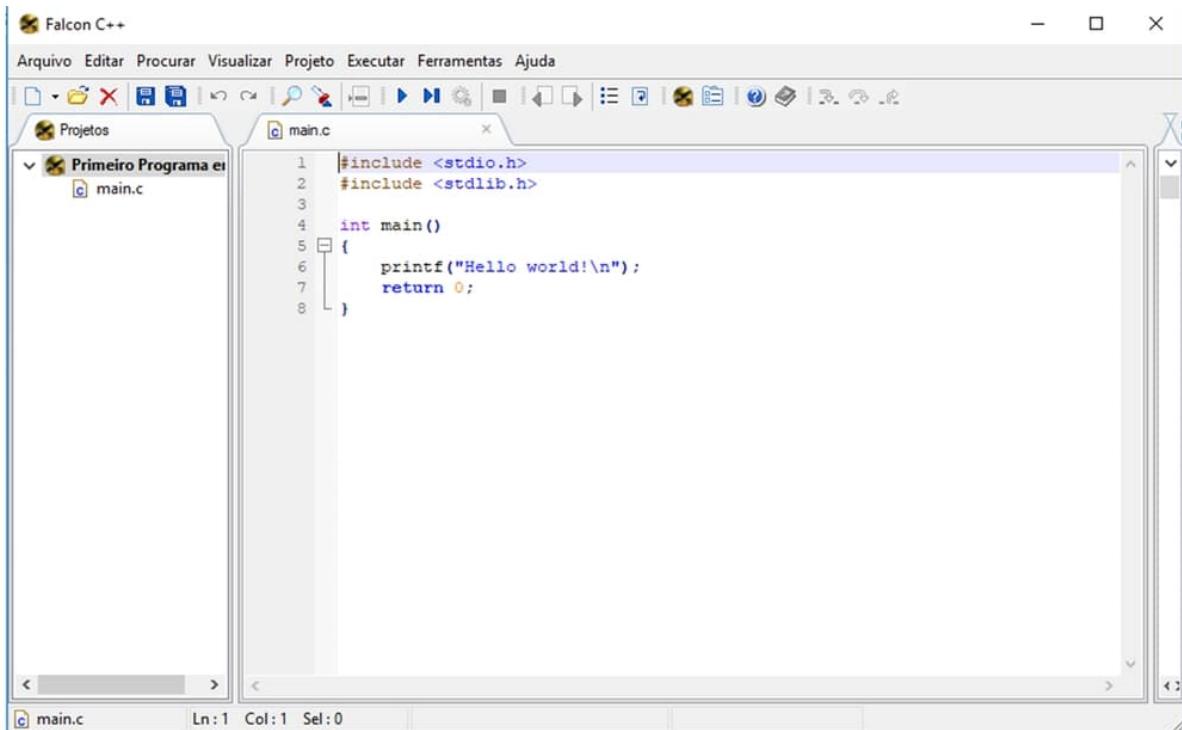


Fonte: elaborado pelo autor.



Ao fim desses passos, o esqueleto de um novo programa C terá sido criado;

Figura 9 - Esqueleto de um Programa no Software Falcon C++.



The screenshot shows the Falcon C++ IDE interface. The menu bar includes Arquivo, Editar, Procurar, Visualizar, Projeto, Executar, Ferramentas, and Ajuda. The toolbar contains various icons for file operations, search, and execution. The left sidebar shows a 'Projetos' tree with 'Primeiro Programa em C' expanded, containing 'main.c'. The main editor window displays the following C code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

The status bar at the bottom indicates Ln:1 Col:1 Sel:0.

Fonte: elaborado pelo autor.



Por fim, podemos utilizar as seguintes opções do menu “**Executar**” para compilar e executar nosso programa:

- **Compilar (Ctrl + F9):** serão compilados todos os arquivos do seu projeto para fazer o processo de “linkagem” com tudo o que é necessário para gerar o executável do seu programa.
- **Compilar e rodar (F9):** além de gerar o executável, essa opção também executa o programa gerado.

Utilizando o Debugger do Falcon C++

- Com o **passar do tempo**, nosso conhecimento sobre programação cresce, assim como a **complexidade de nossos programas**.
- Surge então a necessidade de examinar o nosso programa à procura de erros ou defeitos no código-fonte. Para realizar essa tarefa, contamos com a ajuda de um **depurador** ou **debugger**.
- O **debugger** nada mais é do que um programa de computador usado para testar e depurar (limpar, purificar) outros programas. Entre as principais funcionalidades de um debugger estão:

- A possibilidade de executar um programa **passo a passo**.
- Pausar o programa em pontos predefinidos, chamados pontos de parada ou **breakpoints**, para examinar o estado atual de suas variáveis.

Figura 10 - Utilizando o Debugger no Software Falcon C++.

```
#include <stdio.h>
#include <stdlib.h>
int fatorial (int n) {
    int i, f = 1;
    for (i = 1; i <= n; i++) {
        f = f * i;
    }
    return f;
}

int main() {
    int x, y;
    printf("Digite um valor inteiro: ");
    scanf("%d", &x);
    if(x > 0){
        printf("X eh positivo\n");
        y = fatorial(x);
        printf("Fatorial de X eh %d\n",y);
    } else {
        if(x < 0){
            printf("X eh negativo\n");
        } else {
            printf("X eh zero\n");
        }
    }
    printf("Fim do Programa\n");
    system("pause");
    return 0;
}
```

Fonte: elaborado pelo autor.

- Todas as funcionalidades do debugger podem ser encontradas no menu **Executar**.

1. Primeiramente, vamos colocar dois pontos de parada ou **breakpoints** no programa, nas linhas 16 (if x > 0) e 27 (printf ("Fim do Programa")). Isso pode ser feito clicando no lado direito do número da linha.

Figura 11 - Exemplo de utilização do Debugger no Software Falcon C++.

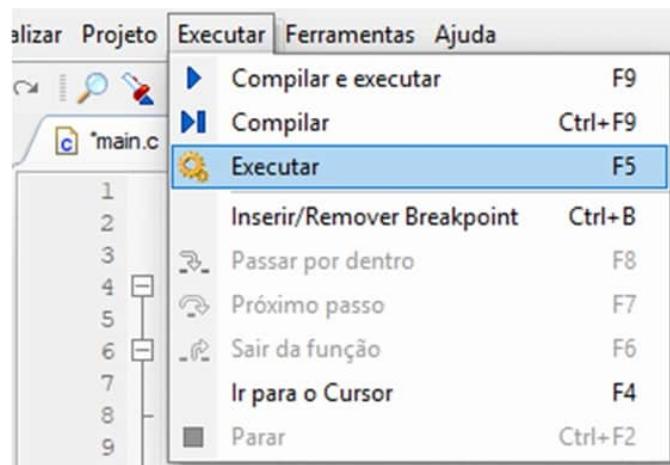
The screenshot shows the Falcon C++ IDE interface with a code editor window titled "main.c". The code implements a factorial function and a main function that prints the factorial of a user-specified integer. Two breakpoints are set: one at line 16 (inside the if(x > 0) block) and another at line 27 (inside the printf("Fim do Programa\n"); line). The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int factorial (int n) {
5     int i, f = 1;
6     for (i = 1; i <= n; i++){
7         f = f * i;
8     }
9     return f;
10 }
11 int main()
12 {
13     int x, y;
14     printf("Digite um valor inteiro: ");
15     scanf("%d", &x);
16     if(x > 0){
17         printf("X eh positivo\n");
18         y = factorial(x);
19         printf("Fatorial de X eh %d\n",y);
20     } else {
21         if(x < 0){
22             printf("X eh negativo\n");
23         } else {
24             printf("X eh zero\n");
25         }
26     }
27     printf("Fim do Programa\n");
28     system("pause");
29     return 0;
30 }
```

Fonte: elaborado pelo autor.

Depois acesse o menu **Executar**, opção **Executar (F5)**;

Figura 12 - Executando o Debugger no Software Falcon C++.



Fonte: elaborado pelo autor.

Figura 13 - Exemplo de utilização do Debugger no Software Falcon C++.

A screenshot of the Falcon C++ IDE during a debugging session. The code editor shows 'main.c' with the following content:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fatorial (int n) {
5     int i, f = 1;
6     for (i = 1; i <= n; i++)
7         f = f * i;
8     return f;
9 }
10
11 int main()
12 {
13     int x, y;
14     printf("Digite um valor inteiro: ");
15     scanf("%d", &x);
16     if(x > 0){
17         printf("X eh positivo\n");
18         Y = fatorial(x);
19         printf("Fatorial de X eh %d\n",y);
20     } else {
21         if(x < 0){
22             printf("X eh negativo\n");
23         } else {
24             printf("X eh zero\n");
25         }
26     }
27     printf("Fim do Programa\n");
28     system("pause");
29     return 0;
30 }
```

The code is annotated with red highlights: the condition 'if(x > 0)' is highlighted, as well as the entire body of the 'main()' function. The status bar at the bottom shows 'Variáveis' (Variables) with 'x = 1' and 'y = 64'.

Fonte: elaborado pelo autor.

- Aperto **F7** para o programa ir para o próximo passo

Figura 14 - Exemplo de utilização do Debugger no Software Falcon C++.

Fonte: elaborado pelo autor.



SAIBA MAIS

1) A primeira versão de C foi criada por Dennis Ritchie, em 1972, nos laboratórios Bell, para ser incluída como um dos softwares a serem distribuídos juntamente com o sistema operacional Unix do computador PDP-11.

2) Linha de comando para compilação de um programa em C:

gcc comando para executar o compilador C;
nomedoarquivo o arquivo deve ser salvo com extensão C e é preciso estar na mesma pasta que ele
-o opção necessária para gerar o executável em C;
nomedoprograma nome do programa que vai ser gerado, se não for digitado, o compilador gera um programa cujo nome será a.out.

Fonte: o autor.



REFLITA

- 1) A linguagem C é uma das mais bem sucedidas linguagens de alto nível já criadas.
- 2) A linguagem C é uma linguagem procedural, ou seja, ela permite que um problema complexo seja facilmente disposto em módulos, sendo cada módulo um problema mais simples.

Fonte: o autor.

Conclusão - Unidade 1

Prezado(a) aluno(a),

Neste capítulo, você estudou sobre a linguagem C, também conhecida como a “linguagem das linguagens”, por ser bastante difundida no meio acadêmico e usada para que você dê os primeiros passos no mundo da programação.

Você entendeu também como surgiu a linguagem por meio do histórico da linguagem C, compreender a estrutura básica da linguagem, além de conhecer o ambiente de desenvolvimento e configurá-lo para que possamos utilizá-los em nossos exemplos.

Nos vemos no próximo capítulo!

Material Complementar



Livro

Linguagem C: Completa e descomplicada

Autor: André Backes

Editora: Elsevier

Sinopse: Criada em 1972 nos laboratórios Bell por Dennis Ritchie, a linguagem C se tornou uma das mais bem sucedidas linguagens de alto nível já criadas, sendo considerada até hoje, na maioria dos cursos de Computação do país, como a linguagem básica para o aprendizado da disciplina. Com um

teor de abstração relativamente elevado, ela está mais próxima da linguagem humana do que do código de máquina. Por isso, é considerada por muitos uma linguagem de difícil aprendizado. Com o objetivo de simplificar o ensino da disciplina, André Backes apresenta neste livro uma nova abordagem que descomplica os conceitos da linguagem por meio de diversos recursos didáticos e ilustrativos, incluindo lembretes e avisos que ressaltem os seus pontos-chave, além de exemplos simples e claros sobre como utilizá-la. Este livro traz um programa de um curso completo de linguagem C, tratando com simplicidade dos assuntos mais complicados até os mais básicos. Livro-texto para estudantes de graduação e pós-graduação em Computação, também pode ser utilizado por profissionais que trabalham com programação e profissionais de áreas não computacionais (biólogos, engenheiros, entre outros) que precisam utilizar o programa em alguma tarefa.

Filme

C tutorial de programação | Aprenda programação C | Linguagem C

Ano: 2015

Sinopse: A Linguagem de Programação C é a linguagem de programação mais popular e a linguagem de programação mais utilizada até agora. É uma linguagem muito simples e elegante.

Referências

ANDRÉ BACKES. **Linguagem C – completa e descomplicada.** Rio de Janeiro, Campus, 2013.

FALCON C++. **Instalação da Ferramenta.** <<http://falconcpp.sourceforge.net/>>. Acesso em abril de 2020.



Centro Universitário Cidade Verde

| Unidade 2

Estrutura de repetição



AUTORIA

Ricardo Bittencourt Socreppa

Introdução

Prezados alunos(as)!

Neste capítulo pretende-se introduzir o conceito de laços de repetição, ou loops, pelo termo em inglês. Laços são recursos que permitem que o programa realize repetições em blocos de instruções indicados ao comando do laço. Com isso cria-se um maior poder para os códigos criados pelo programador, permitindo a combinação de laços com outros comandos da linguagem. Dessa forma, o texto irá abordar inicialmente como os laços funcionam e integram a lógica de programação, apresentando a estrutura básica do comando while.

O capítulo continuará apresentando uma outra forma de laço com o comando for, que tem uma estrutura de repetição um pouco diferente do comando while. E por fim a combinação de condições com os comandos break e continue, que permitem a quebra ou continuidade da repetição.

Pretende-se também introduzir o entrada e saída de dados. A maioria dos programas necessita de uma interação do usuário de forma a criar interatividade. Sendo assim, neste capítulo será mostrado também como trabalhar com o input do usuário, para ter interação com o usuário e ter como parte dos programas o cálculo a partir da entrada de dados. O capítulo finalizará apresentando os comentários, estruturas auxiliares ao código que ajudam no seu entendimento, que são ignorados pelo interpretador, isto é, ele não é executado, mas ajuda no entendimento do código.

Venha comigo, te espero lá!



Centro Universitário Cidade Verde

Laços



AUTORIA

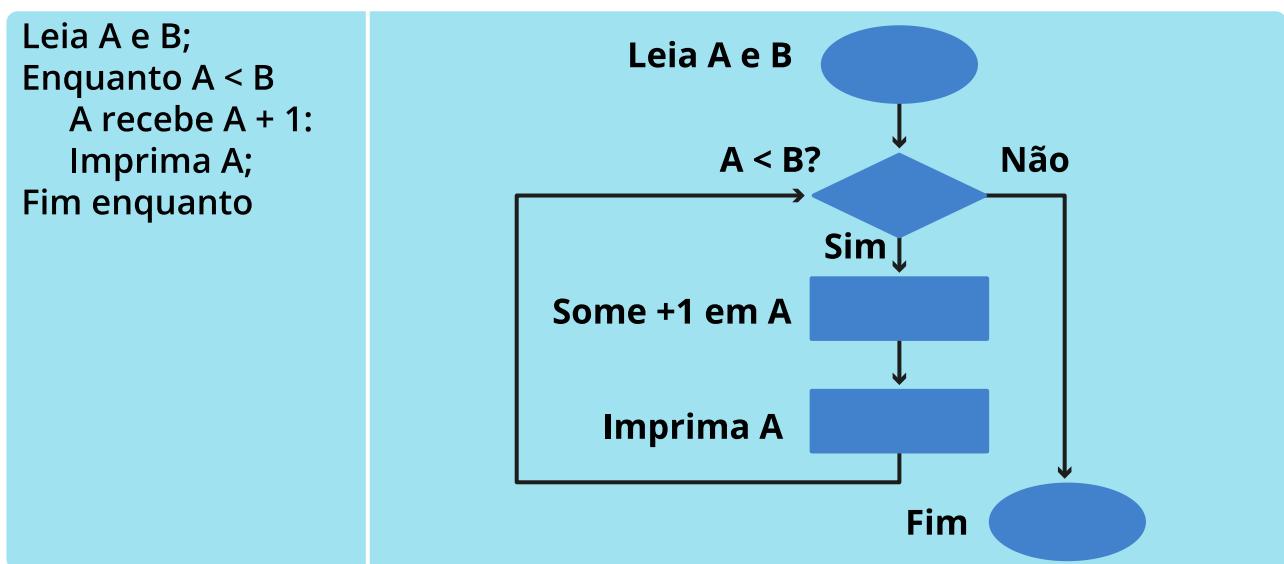
Ricardo Bittencourt Socreppa

Esta seção tem como finalidade apresentar ao leitor o conceito de laços, de forma a continuar a criação da base para o entendimento de códigos mais complexos. Uma das vantagens dos laços é que a tarefa de repetir instruções não fica mais a cargo do programador, e sim do programa, que passa a repetir instruções de acordo com o código.

Existem casos em que é preciso que um bloco de comandos seja executado mais de uma vez se determinada condição for verdadeira.

Para isso, precisamos de uma estrutura de repetição que permita executar um conjunto de comando quantas vezes forem necessárias. Pode-se ver esse fluxograma na Figura 1.

Figura 1 - Exemplo simples de pseudocódigo e fluxograma.



Fonte: Adaptado de BACKES (2013).

Condição

Qualquer **expressão relacional** que resulte em uma resposta **verdadeiro** ou **falso**.

- Operadores usados:

- **Matemáticos:** +, -, *, /, %
- **Relacionais:** >, <, >=, <=, ==, !=
- **Lógicos:** &&, ||

- Na execução a condição é avaliada:

- Se a condição for **verdadeira**, a sequência de comandos **será executada**;
- Se a condição for **falsa**, a sequência de comandos **não** será **executada**;

Laço infinito

Um **laço infinito** é uma sequência de comandos que **nunca termina**.

- Ocorre por uma **erro** de **programação**:

- **Não** é **definido** uma condição de **parada**
- A condição de parada existe, mas **nunca** é **atingida**

Veja um exemplo a seguir Exemplo:

```
x recebe 4
enquanto x < 5 faça
    x recebe x - 1;
    imprima x;
fim enquanto
```

Comando while

O comando **while** equivale ao comando “**enquanto**” no pseudocódigo apresentado anteriormente.

- Sua forma geral é:

```
while (condição) {
    sequência de comandos;
}
```

- Avaliação da condição:

- Se for **verdadeira** ou possuir valor **diferente** de **0**, a sequência de comandos é executada e o fluxo de comando será desviado para um novo teste de condição.
- Se for **falsa** ou possuir um valor **igual** a **0**, a sequência de comandos não será executada;

- Vejamos no Código 1 um exemplo em C do comando **while**:

Código 1. Código em C da utilização do comando while.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a, b;
    printf("Digite o valor de a: ");
    scanf("%d",&a);
    printf("Digite o valor de b: ");
    scanf("%d",&b);
    while (a < b)
    {
        a = a + 1;
        printf("%d \n", a);
    }
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.



AUTORIA
Ricardo Bittencourt Socreppa

UniFCV Centro Universitário Cidade Verde

Um tipo especial de laço com o for

Um tipo especial de laço com o for

O comando **while** repete um laço de código enquanto uma condição for True, mas se o programador quisesse que o código repetisse um bloco de instruções um certo número de vezes, ele teria que declarar uma variável, verificar se essa variável ultrapassou um certo valor, e incrementar essa variável.

Para facilitar esse processo, existe um outro comando de repetição chamado de **for**, que teria tradução de para. O comando **for** trabalha com uma sequência de algum tipo, como uma string, lista, ou dicionários que serão vistos mais adiante.

Sequência do comando for:

1. **Inicialização:** as **variáveis recebem** um **valor inicial** para usar dentro do for

2. **Condição:**

- Se for **verdadeira** ou **diferente** de **0**, a sequência de comando é executado e desviado para o **incremento**;
- Se for **falsa** ou **igual** a **0**, a sequência de comandos não será executado (fim do for);

3. **Incremento**

- Faz o **incremento/decremento** das **variáveis** utilizadas no **for** e desviado para a condição

Vejamos abaixo um exemplo de código em C utilizando o comando **for**:

Código 2 - Código em C da utilização do comando for.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a, b, c;
    printf("Digite o valor de a: ");
    scanf("%d",&a);
    printf("Digite o valor de b: ");
    scanf("%d",&b);
    for (c = a; c <= b; c++)
    {
        printf("%d \n", c);
    }
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

Pode-se verificar que, apesar de ter um funcionamento de repetição parecido com o while, o for trabalha com sequências em vez de condições. A condição dele está implícita no comando, verificando somente se a sequência a ser usada já chegou ao fim.

O comando for primeiro cria a sequência “sequencia” e coloca o primeiro item dela na variável “elemento” e checa se a “sequencia” está vazia ou se chegou ao fim dela, para somente depois executar as instruções que estão endentadas. Depois que essas instruções são executadas, o código repete o processo, só que dessa vez ela seleciona o segundo item da sequência, até que não haja mais itens na sequência.

Aninhamento de repetições

Aninhamento de repetições nada mais é que **um comando de repetição dentro do outro**;

Forma geral:

```
repetição (condição 1) {
    sequência de comandos;
    repetição (condição 2) {
        sequência de comandos;
    }
}
```

Vejamos no Código 3 um exemplo aninhamento de repetições:

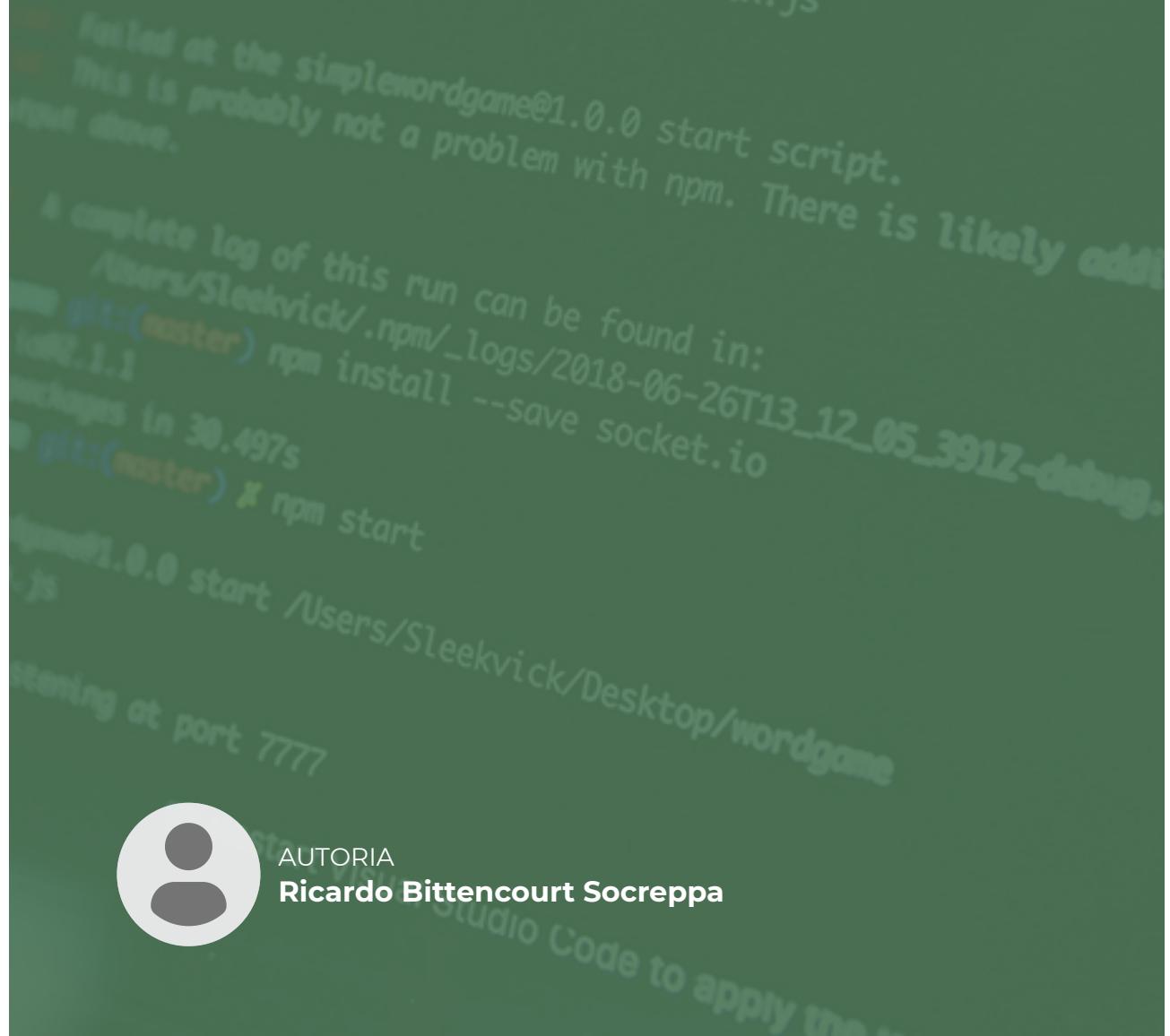
Código 3 - Código em C da utilização de um aninhamento de repetições.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i = 1, j;
    while(i < 5) {
        j = 1;
        while(j < 5) {
            if(i == j){
                printf("1 ");
            } else {
                printf("0 ");
            }
            j++;
        }
        printf("\n");
        i++;
    }
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.



Usando break e continue para modificar a execução do laço



Até o momento vimos como realizar uma repetição até que a condição inicial do laço falhe. Com o uso dos comandos break, o programador pode alterar esse fluxo de execução dos testes de condição, fazendo com que o código saia da estrutura de repetição (do código indentado a ele), ou apenas pule as instruções seguintes e passe para a próxima interação do fluxo.

Observe no Código 4 a utilização do comando break:

Código 4 - Código em C da utilização de um break.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int a, b;
    printf("Digite o valor de a: ");
    scanf("%d", &a);
    printf("Digite o valor de b: ");
    scanf("%d", &b);
    while(a <= b) {
        a++;
        if(a == 5){
            break;
        }
        printf("%d \n", a);
    }
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.



Entrada e Saída de Dados



AUTORIA

Ricardo Bittencourt Socreppa

Saída de Dados

Escrevendo variáveis na tela:

- Segundo BACKES (2013), a função **printf()** é uma das funções de saída/ escrita de dados da linguagem C.
- Seu nome vem da expressão em inglês **print formatted**, ou seja, escrita formatada.
- A forma geral da função **printf()** é:

- *printf(" tipos de saída", lista de variáveis).*

- A função **printf()** recebe dois parâmetros de entrada:

- **“tipos de saída”**: conjunto de caracteres que especifica o formato dos dados a serem escritos e/ ou o texto a ser escrito.
- **lista de variáveis**: conjunto de nomes de variáveis, separados por vírgula, que serão escritos.

- Também pode ser usada quando queremos escrever apenas uma mensagem de texto simples na tela:

Código 5 - Código em C da utilização do printf().

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Esse texto sera escrito na tela");
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

Tipos de Saída

Quadro 1 - Quadro demonstrativo dos tipos de saída.

| Comando | Descrição |
|----------|--|
| %c | Escrita de um caracter (char) |
| %d ou %i | Escrita de números inteiros (int ou char) |
| %u | Escrita de números inteiros sem sinal (unsigned) |
| %f | Escrita de números reais (float ou double) |
| %s | Escrita de vários caracteres |
| %p | Escrita do endereço de memória |
| %e ou %E | Escrita em notação científica |

Fonte: elaborada pelo autor.

Escrevendo valores formatados

Quando queremos escrever dados formatados na tela usamos a forma geral da função printf();

Cada tipo de saída é precedido por um sinal de %, e um tipo de saída deve ser especificado para cada variável a ser escrita.

Código 6 - Código em C da mostrando como escrever valores formatados.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int x = 10;
    printf("%d \n", x);
    float y = 5.0;
    printf("%d %f \n", x, y);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

Entrada de Dados

Lendo variáveis do teclado

- Segundo BACKES (2013), a função scanf() é uma das funções de entrada/ leitura de dados da linguagem C.
- Seu nome vem da expressão em inglês scan formatted, ou seja, leitura formatada.
- Basicamente, lê do teclado um conjunto de valores, caracteres e/ ou sequência de caracteres de acordo com o formato especificado.
- A forma geral da função scanf() é:

- `scanf(" tipos de entrada", lista de variáveis).`

- A função `scanf()` recebe dois parâmetros de entrada:

- “tipos de entrada”: conjunto de caracteres que especifica o formato dos dados a serem lidos.
- lista de variáveis: conjunto de nomes de variáveis que serão lidos e separados por vírgula, em que cada nome de variável é precedido pelo operador &.

Tipos de Entrada

Quadro 2 - Quadro demonstrativo dos tipos de entrada.

| Comando | Descrição |
|----------|--|
| %c | Leitura de um caractere (char) |
| %d ou %i | Leitura de números inteiros (int ou char) |
| %f | Leitura de números reais (float ou double) |
| %s | Leitura de vários caracteres |

Fonte: elaborada pelo autor.

Lendo variáveis do teclado

O comando `scanf()` permite receber dados formatos. Veja no Código 7 uma utilização da função:

Código 7 - Código em C da mostrando como ler variáveis do teclado.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x, z;
    float y;
    scanf("%d", &x);
    scanf("%f", &y);
    scanf("%d%f", &x, &y);
    scanf("%d %d", &x, &z);
    scanf("%d %d %f", &x, &z, &y);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.



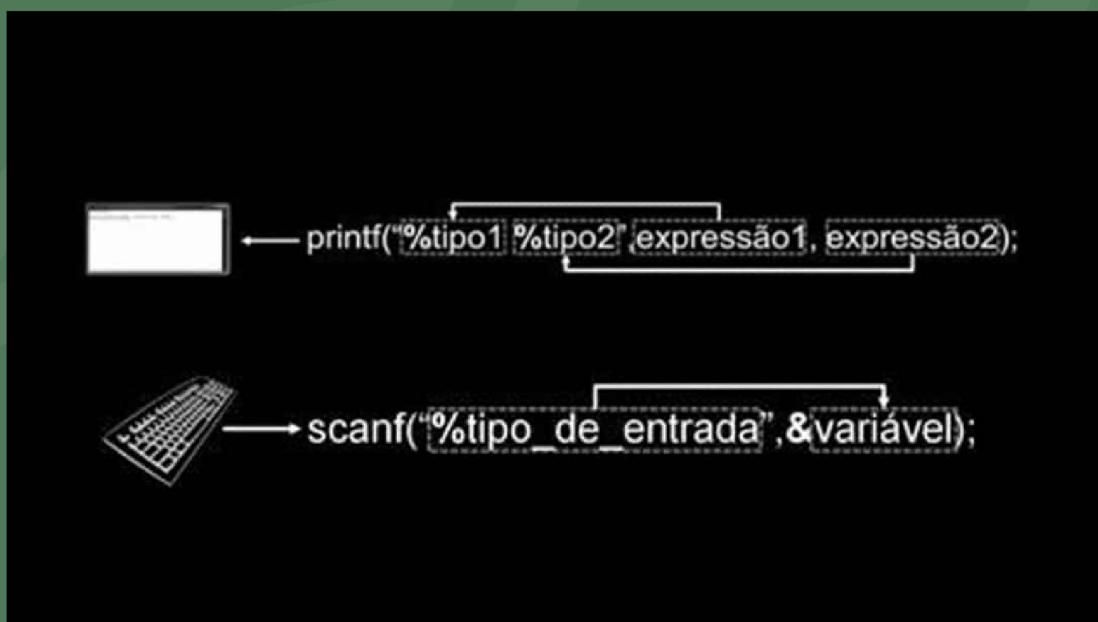
SAIBA MAIS

1) Quando queremos escrever dados formatados na tela usamos a forma geral da função printf() ou scanf():



Fonte: BACKES (2013)

2) Cada tipo de saída é precedido por um sinal de %, e um tipo de saída deve ser especificado para cada variável a ser escrita:



Fonte: BACKES (2013)





REFLITA

Programação imperativa

O paradigma imperativo apoia-se na base teórica proporcionada pela Máquina de Turing, tendo como principal lastro tecnológico, a arquitetura de von Neumann. Essa arquitetura leva os programas a terem como recurso central de armazenamento de informação valores armazenados em memória, em forma de estruturas de dados, ou seja, agrupamento de variáveis. As instruções do programa costumam também ser organizadas em posições lógicas contíguas de memória, o que pode tornar mais eficiente o processamento (PELEGRINI, 2009).

Nesse paradigma, a ideia central é o conceito de estado de um programa, materializado na configuração da memória do programa e dos seus dados, e que é alterado durante a execução do mesmo através de sucessivas modificações dos valores das variáveis, impostas pelas instruções do programa sobre o seu estado.

Essa função é desempenhada, principalmente, pelos comandos de atribuição, que alteraram o estado de um programa através da substituição de um valor, contido em uma posição de memória, por algum outro valor.

Quando uma linguagem é capaz de fornecer recursos adequados que permitam a implementação de qualquer algoritmo que possa ser projetado, essa linguagem se diz Turing-Completa. Dessa forma, uma linguagem de programação imperativa que disponibilize variáveis e valores inteiros, as operações aritméticas básicas, comandos de atribuição, comandos condicionais e interativos é considerada Turing-completa (SEBESTA, 2010).

Além dos comandos de atribuição, as linguagens de programação imperativas costumam disponibilizar ao programador: declarações de variáveis, expressões, comandos condicionais, comandos iterativos e abstrações procedimentais.

A abstração procedural dá ao programador a possibilidade de atentar para as relações existentes entre um procedimento e a operação que ele realiza (em particular, entre uma função e o cálculo que ela executa) sem preocupações acerca da maneira como essas operações são realizadas.

O refinamento gradual de abstrações é uma maneira sistemática muito empregada no desenvolvimento de programas e de muitas outras modalidades de sistemas computacionais (técnica dos refinamentos sucessivos). Usando abstração procedural, um programador pode partitionar a lógica de uma função, idealizada de forma macroscópica, em um grupo de funções mutuamente dependentes, mais simples e/ou mais específicas.

Com o fim de simplificar o desenvolvimento de algoritmos complexos, as linguagens imperativas modernas oferecem ao programador suporte a matrizes e estruturas de registro, além de bibliotecas extensíveis de funções, que, facilitando o reaproveitamento de partes já desenvolvidas de programas, evitam que operações comuns necessitem ser reprogramadas, como é o caso de operações de entrada e saída de dados, gerenciamento de memória, manipulação de cadeias, estruturas de dados clássicas, etc.

São exemplos de linguagens imperativas, entre inúmeras outras, as linguagens FORTRAN e C.

[...]

Fonte: SILVA, 2011, p. 30-34



Conclusão - Unidade 2

Prezado(a) aluno(a),

Neste capítulo introduzimos o conceito de laços de repetição, ou loops pelo termo em inglês. Vimos que Laços são recursos que permitem que o programa realize repetições em blocos de instruções edentados ao comando do laço. Com isso podemos criar um maior poder para os códigos criados pelo programador, permitindo a combinação de laços com outros comandos da linguagem.

Dessa forma, o texto abordou como os laços funcionam e integram a lógica de programação, apresentando a estrutura básica do comando while. O capítulo continuou apresentando uma outra forma de laço com o comando for, que tem uma estrutura de repetição um pouco diferente do comando while. E por fim mostrou o comando break, que permite a quebra da repetição.

E para finalizar, vimos também sobre a entrada e saída de dados. Entendemos que a maioria dos programas necessita de uma interação do usuário de forma a criar interatividade. Sendo assim, neste capítulo foi mostrado como trabalhar com o input do usuário, para ter interação com o usuário e ter como parte dos programas a partir da entrada de dados.

Nos vemos no próximo capítulo, forte Abraço!



Livro

Introdução à Programação com a Linguagem C: Aprenda a Resolver Problemas com uma Abordagem Prática

Autor: Rodrigo de Barros Paes

Editora: Novatec

Sinopse: Este livro oferece conteúdo abrangente e plenamente compatível para ser utilizado como material didático em disciplinas introdutórias de programação, seja no ensino médio, cursos técnicos, universidades ou mesmo em cursos de curta duração. O texto é baseado em uma metodologia de aprendizado por experiência e com grande foco na prática de exercícios. Esse aprendizado se dá por meio de quatro etapas cuidadosamente exploradas em cada assunto: experiência concreta, pequenas modificações, analogia e experimentação livre. Diferentemente do que é comum encontrar nos livros de programação, neste livro os conceitos são introduzidos sempre que se fazem necessários para resolver um determinado problema. Ou seja, primeiro apresenta-se um problema prático para ser resolvido e só então os conteúdos necessários são introduzidos.

Os leitores terão à disposição uma ferramenta on-line para a correção automática dos exercícios propostos. O escopo do livro abrange desde o início, com o entendimento sobre o que são algoritmos e o funcionamento básico de um computador, até questões mais avançadas, como recursão, alocação dinâmica, ponteiros e várias dicas de programação. Ao final do livro, o leitor estará apto a resolver problemas utilizando a linguagem de programação C.

Filme

Programação em C Aula 1 Abertura do Curso de C

Ano: 2014

Sinopse: Como programar em C? Assista a essa primeira aula e comece a aprender desde já! Então, seja bem-vindo à primeira aula do nosso Curso C! Estamos, gradativamente, disponibilizando um curso completo de programação à

linguagem C. Nosso objetivo é ensinar-lhe detalhadamente, sem enrolações e sem investir muito tempo com aspectos da linguagem, considerados irrelevantes.

Esse curso foi projetado para auxiliar (ou servir como guia) a estudantes que não conhecem absolutamente nada de programação, como também, a estudantes que já possuem alguma noção de lógica.

A lógica de programação é abordada durante o curso, então, não há necessidade de qualquer conhecimento prévio para poder iniciar os estudos!!

Referências

ANDRÉ BACKES. **Linguagem C – completa e descomplicada.** Rio de Janeiro, Campus, 2013.

SILVA, S. R. B. **Software adaptativo: método de projeto, representação gráfica e implementação de linguagem de programação.** Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de mestre em Engenharia Elétrica. São Paulo, 2011.



| Unidade 3

Modularização do sistema por meio de funções



AUTORIA

Ricardo Bittencourt Socreppa

Introdução

Prezados alunos(as)!

Neste capítulo será introduzido o conhecimento sobre os comentários e funções. Você já pode usar funções do sistema, passando ou não atributos.

Este capítulo proverá os meios para que o você possa criar suas próprias funções e ver as vantagens de fazer isso. Uma delas é o aproveitamento do código, que permite ao programador reutilizar sequências de código, e separar também as sequências de código, que o torna mais legível e portável.

Este capítulo se concentra em apresentar as principais características e vantagens das funções e apresenta o uso de funções com argumentos e como utilizar os argumentos padrões.

Ótimos estudos!

Comentando seu código



AUTORIA

Ricardo Bittencourt Socreppa

Seguindo na mesma linha de BACKES (2013), o comentário é um **trecho de texto** incluído dentro do programa para **descrever alguma coisa**, por exemplo, o que aquele pedaço do programa faz. Permite fazer a **documentação interna** de um programa.

A linguagem C permite fazer comentários de duas maneiras diferentes:

Por linha

- basta adicionar **//** na frente da linha.

Pro bloco.

- basta adicionar **/*** no começo da primeira linha de comentário e ***/** no final da última linha de comentário.

Código 1 - Exemplos de como comentar um código-fonte.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    /* A função printf
       serve para imprimir
       na tela */
    printf("Hello world!\n");
    // Faz uma pausa no programa
    system("pause");
    return 0;
}
```

Fonte: elaborado pelo autor.



Centro Universitário Cidade Verde

Funções



AUTORIA

Ricardo Bittencourt Socreppa

Os códigos desenvolvidos até agora foram pequenos, mas caso seja necessário criar algo mais robusto, sem funções, será difícil, caso se queira repetir o código em outra parte do programa poderia se copiar as linhas, mas caso tenha de mudar algo nele, o programador teria que refazer em vários trechos do programa. Para esses casos o melhor é usar funções. Isso é colocado como reusabilidade do código. Dentre as vantagens de reutilizar o código, tem-se: redução do tempo de desenvolvimento, redução dos erros de programação, aumento da estabilidade do programa, compartilhamento de código, facilidade de entendimento do código e aumento da eficiência do programa.

Esta seção concentrará em mostrar as funções. As funções são comuns em Python e em outras linguagens de programação. Elas podem ser definidas como agrupamentos de partes de códigos, que podem ser chamadas quando o programador precisar executar o trecho do código. Sendo assim, funções podem ser vistas como miniprogramas dentro do programa.

As funções possuem um nome, que é a forma que ela será chamada para executar o trecho de código definido na função. Algumas são incluídas com os módulos do Python, como o print. Um módulo é um arquivo que contém definições de funções e classes (que infelizmente está fora do escopo desta obra, mas você terá oportunidade de aprender futuramente durante o curso). Dessa forma, o programador pode criar e compartilhar suas funções dentro de um arquivo.

Para definir uma função, usa-se o comando def, seguindo do nome da função e dentro de parênteses, se coloca os argumentos, ou nada caso a função não tenha argumentos. Uma função tem o formato como mostra no Código 2.

Código 2. Exemplo de declaração de uma função.

```
tipo_retornado nome_funcao (lista de parametros) {  
    sequência de comandos  
}
```

Fonte: elaborada pelo autor.

Observações:

- A declaração de ser feita antes do código no qual é utilizada, ou seja, antes da clausula main().
- Pode ser declarada depois da main() neste caso precisamos declarar o protótipo da função antes da main().

O trecho de código no Código 3 apresenta a declaração antes da main():

Código 3 - Exemplo de declaração antes da main().

```
#include <stdio.h>
#include <stdlib.h>
int quadrado(int a) {
    return a * a;
}
int main() {
    int n1, n2;
    printf("Digite um numero: ");
    scanf("%d",&n1);
    n2 = quadrado(n1);
    printf("O quadrado de %d eh %d\n",n1,n2);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

O trecho de código no Código 4 apresenta a declaração depois da main():

Código 4 - Exemplo de declaração depois da main().

```
#include <stdio.h>
#include <stdlib.h>
// protótipo da função
int quadrado(int a);
int main() {
    int n1, n2;
    printf("Digite um numero: ");
    scanf("%d",&n1);
    n2 = quadrado(n1);
    printf("O quadrado de %d eh %d\n",n1,n2);
    system("pause");
    return 0;
}
int quadrado(int a) {
    return a * a;
}
```

Fonte: elaborada pelo autor.

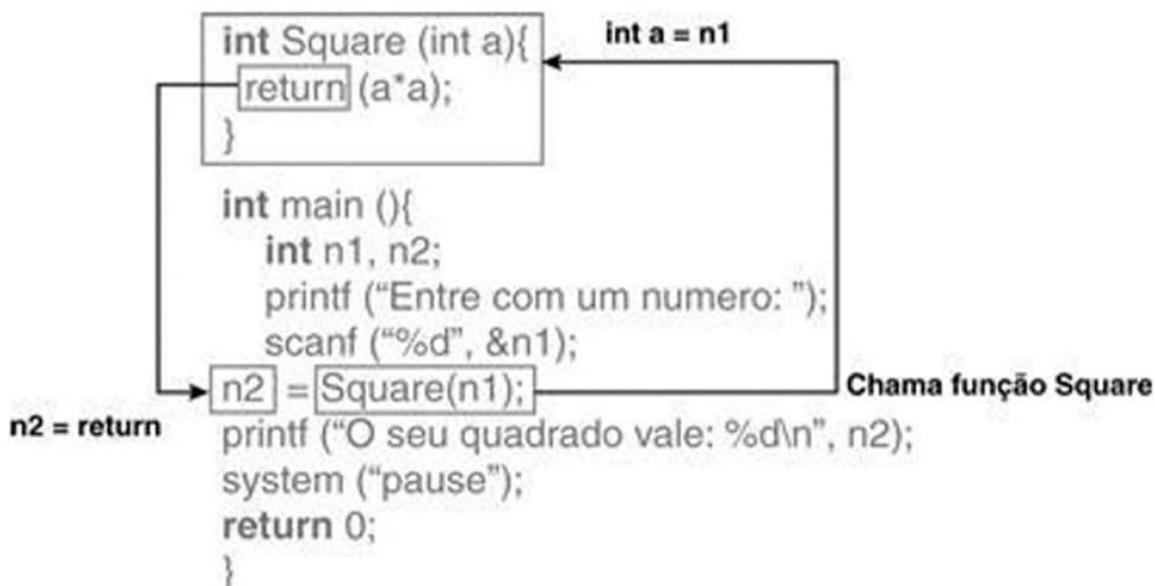
Funcionamento da Função

Independente da função ela segue os **seguintes passos**:

1. O código do programa é executado **até encontrar** a chamada da **função**;
2. O **programa** é **interrompido temporariamente**, para executar a função
3. Se **houver parâmetro** na função os **valores são copiados** para ela.
4. **Comandos** da **função** são **executados**
5. Quando a **função termina** ou **encontra** a palavra **return**, o **programa volta ao ponto** que foi **interrompido**
6. Se houver um **valor** no **return** ele volta para a **variável** que foi escolhida para **receber o valor**;

Veja abaixo no Código 5 um exemplo de Funcionamento de uma Função:

Código 5 - Exemplo de funcionamento de uma função.



Fonte: BACKES (2013).

Parâmetros de uma função

Basicamente são **valores** que **passados** com o **objetivo** de ser **utilizado** dentro da **função**. O parâmetros devem ser **separados** por “,”.

Veja o exemplo no Código 6 com é, de uma forma geral, a passagem de parâmetros de uma função:

Código 6 - Forma geral de passagem de parâmetros de uma função.

```
tipo_retornado nome_funcao (tipo nome, tipo nome2, tipo nome3,  
...) {  
  
    sequência de comandos  
  
}
```

Fonte: elaborada pelo autor.

Funções sem parâmetros

- Dependendo da função, ela pode **não possuir parâmetros**.
- Declaração:

Deixar a lista de parâmetros em branco:

- void imprime()

Colocar void entre parênteses:

- void improme (void)

Veja no Código 7 que sem declarar o **void**, **não** é possível colocar **valor de parâmetro** na **função**:

Código 7 - Exemplo valor de parâmetro na função.

```
#include <stdio.h>
#include <stdlib.h>
int imprime(){
    printf("Teste da Funcao \n");
}
int main() {
    imprime();
    imprime(5);
    imprime(5, 'a');
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int imprime(void){
    printf("Teste da Funcao \n");
}
int main() {
    imprime();
    imprime(5);
    imprime(5, 'a');
    return 0;
}
```

Fonte: elaborada pelo autor.

Retorno da função

É a **maneira** como a **função devolve o resultado** (se existir) para quem a chamou.

Veja abaixo no Código 8 como é dado a forma geral de uma função:

Código 8 - Forma geral de uma função.

```
tipo_retornado nome_funcao (lista de parametros) {
    sequência de comandos
}
```

Fonte: elaborada pelo autor.

Perceba que o **tipo_retornado** estabelece o tipo do valor que a função vai devolver.

Tipos de retorno:

- **Básicos**: int, float, char, double, void e ponteiros.
- Tipos definidos pelo usuário: struct, array.

Função sem retorno (void)

A seguir, no Código 9 um exemplo de função sem retorno (void):

Código 9 - Função sem retorno (void).

```
#include <stdio.h>
#include <stdlib.h>
void imprime(int a) {
    int i;
    for(i = 0; i <= a; i++) {
        printf("Linha %d \n", i);
    }
}
int main() {
    imprime(5);
    return 0;
}
```

Fonte: elaborada pelo autor.

Função com retorno

O comando **return** é utilizado para retorna o valor da função.

Forma Geral:

- **return expressao;**

Observações importantes:

- A **expressão** deve ser **compatível** com o **tipo de retorno declarado na função**.
- Quando **chega** no comando **return** a **função** é **encerrada imediatamente**.

A seguir, no Código 10 um exemplo de função com retorno:

Código 10 - Função com retorno.

```
#include <stdio.h>
#include <stdlib.h>
int soma(int x, int y) {
    return x + y;
}
int main() {
    int a,b;
    printf("Digite a:");
    scanf("%d",&a);
    printf("Digite b:");
    scanf("%d",&b);
    printf("A soma eh %d",soma(a,b));
    return 0;
}
```

Fonte: elaborada pelo autor.

Funções com múltiplos argumentos

```
untitled          — default.htm      — index.htm      — page.htm      — post.htm
23      <script>window.addEventListener('load',function(e){var r=document.createElement('script');r.parentNode.insertBefore(r,e.target);ga('create','UA-61868113-3');ga('send','pageview');});</script>
24
25
26
27
28      </head>
29      <body class="({body_class})">
30
31          <a href="http://portfolio.          id="docs">View My Portfolio</a>
32
33          <section id="wrapper">
34              <header id="header">
35                  <a id="title" class="index" href="/">
36                      
37                      <h1>{{blog.title}}</h1>
38                  </a>
39                  <p class="header-description">Development Lead at <a href="http://elementthree.com" target="_blank">El-        ment Three</a>, <small>Co-Creator of <a href="http://zachphillips.com">Hello</a></small> or <a href="mailto:tellellizachphillips.com">tellellizachphillips.com</a><br/><br/><a href="http://twitter.com/zachphillips" target="_blank"><img class="icon-social-twitter" alt="Twitter icon" /></a>
40                      <a href="http://zachphillips.com" target="_blank"><img class="icon-social-linkedin" alt="LinkedIn icon" /></a>
41                      <a href="https://github.com/zachphillips" target="_blank"><img class="icon-social-github" alt="GitHub icon" /></a>
42
43          </header>
44
45          <div id="content">
46              <div id="post">
47                  <h2>{{post.title}}</h2>
48                  {{post.content}}
49
50                  <div id="comment">
51                      <h3>Comments</h3>
52
53                      <ul>
54                          <li>{{comment.name}} - {{comment.date}}</li>
55                      </ul>
56
57                      <form>
58                          <input type="text" placeholder="Type your comment here..."/>
59
60                          <button type="submit">Post Comment</button>
61
62                      </form>
63
64                  </div>
65
66          </div>
67
68      </div>
69
70      <script>document.getElementById('post').style.height = document.getElementById('comment').offsetHeight + 100 + 'px';</script>
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86      <footer id="footer">
```



AUTORIA

Ricardo Bittencourt Socreppa

Na linguagem C podemos passar 4 tipos de parâmetros em uma função, são elas:

1. Passagem de Valor
2. Passagem por Referência
3. Passagem de Arrays
4. Passagem de Estruturas

Passagem de Valor

Este tipo de passagem é o **padrão** para todos os **tipos básicos**:

- int
- float
- double
- Char

Qualquer **modificação** que a função fizer nos **parâmetros** ocorre **somente dentro** da **função**.

No Código 11 segue um exemplo de função com passagem de valor:

Código 11 - Função com passagem de valor.

```
#include <stdio.h>
#include <stdlib.h>
void soma_mais_um(int n) {
    n++;
    printf("Dentro da Função %d \n", n);
}
int main() {
    int x = 5;
    printf("Antes da Função %d \n", x);
    soma_mais_um(x);
    printf("Depois da Função %d \n", x);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

Passagem por Referência

- É quando o **valor passado** por **parâmetros** a **função modifica** o **valor original passado** a ela.
- Exemplo:
scanf();
- Na passagem de parâmetros por referência, **não se passam valores** das **variáveis, mas os endereços** da **variável** na **memória**.
- Para **passar** um **parâmetro** por **referência**, usa-se o **operador “*” na frente do nome** do **parâmetro** na declaração da função.
- Na **chamada da função**, é necessário utilizar o **operador “&” na frente do nome da variável** que será passada por referência.

No Código 12 segue um exemplo de função com passagem por referência:

Código 12 - Função com passagem por referência.

```
#include <stdio.h>
#include <stdlib.h>
void soma_mais_um(int *n) {
    *n = *n + 1;
    printf("Dentro da Função %d \n", *n);
}
int main() {
    int x = 5;
    printf("Antes da Função %d \n", x);
    soma_mais_um(&x);
    printf("Depois da Função %d \n", x);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

Passagem de Arrays

Para passar um array como parâmetro, além de **passar o array**, é necessário passar uma **segunda variável** informando o **tamanho** do **array** que está sendo **passado**.

Os array são **sempre** passados por **referência**

A **passagem** por **referência** evita **cópia desnecessária** de **grande quantidades de dados** para **outras áreas da memória**, afetando o desempenho do programa

Forma Geral:

```
void imprime (int *m, int n);
void imprime (int m[], int n);
void imprime (int m[5], int n);
```

Na chamada da função **passamos o nome do array sem colchetes, sem índice e sem operador “&”**

No Código 13 segue um segue de função com passagem de arrays:

Código 13 - Função com com passagem de arrays.

```
#include <stdio.h>
#include <stdlib.h>
void imprime(int *n, int m) {
    int i;
    for (i = 0; i < m; i++) {
        printf("%d \n", n[i]);
    }
}
int main() {
    int v[5] = {1,2,3,4,5};
    imprime(v,5);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.

Passagem de Estruturas

- Como já vimos anteriormente, estruturas são como **um conjunto de variáveis**.
- Pode-se passar por duas formas distintas:

Toda a estrutura

- Basta declarar na lista de parâmetros a estrutura.

Apenas determinados campos da estrutura

- Declarando uma variável do mesmo tipo da variável da estrutura.

No Código 14 segue um exemplo de função com passagem de estrutura:

Código 14 - Função com passagem de estruturas.

```
#include <stdio.h>
#include <stdlib.h>
struct ponto {
    int x,y;
};
void imprime(int p) {
    printf("Valor = %d\n",p);
}
int main() {
    struct ponto p1 = {10,20};
    imprime(p1.x); imprime(p1.y);
    system("pause");
    return 0;
}
```

Fonte: elaborada pelo autor.



SAIBA MAIS

A visão da função na metodologia de programação

Começando com uma função geral e abstrata do que o programa deve fazer, este é dividido em várias funções também abstratas. Qualquer uma destas funções pode, de forma hierárquica, ser dividida em mais funções abstratas e assim sucessivamente. A metodologia termina por chegar a um nível tal de formalidade no qual a função geral pode ser implementada no computador.

As funções aceitam entradas e saídas de forma a generalizar o processo aplicando-se a quaisquer dados de tipos determinados. As entradas são os argumentos e dados globais usados no procedimento ou função. As saídas são o valor de retorno (as funções podem ou não possuir valor de retorno), modificações feitas através de ponteiros e referências, bem como mudanças em dados globais (Figura abaixo).



As entradas e as saídas desempenham o papel da interface para as funções. O usuário só precisa entender a interface para fazer uso das funções. A sequência de funções é um detalhe oculto. Por isso, e de forma geral, consideramos uma função como um único elemento abstrato.

Tomemos como exemplo a função `Ordena` cuja entrada é uma lista de três parâmetros inteiros e cuja saída é a mesma lista em ordem crescente. Assim, se $x=7$, $y=2$ e $z=4$ teremos, após executar o

procedimento Ordena (x,y,z), $x=-2$, $y=4$ e $z=7$. Outro exemplo é a função Maior cuja entrada são dois parâmetros reais e cuja saída é o valor de retorno maior entre os dois. Assim Maior (3,4)=4, Maior(7,-2)=7, etc.

Fonte: BARROS, 2006.



REFLITA

A complexidade de um programa pode ser dominada através da Programação Estruturada, uma metodologia que se compõe de quatro princípios básicos:

1. Princípio da abstração é a concepção ou visão do programa separado de sua realidade. É a simplificação de fatos, descrevendo o que está sendo feito sem explicar como está sendo feito;
2. Princípio da formalidade possibilita analisar os programas de forma matemática. Fornece uma abordagem rigorosa e metódica. Possibilita a transmissão de ideias e instruções sem ambiguidades e permite que estas sejam automatizadas;
3. Princípio da divisão é a subdivisão organizacional de um programa em um conjunto de partes menores e independentes, mais fáceis de serem entendidas, resolvidas, manipuladas e testadas individualmente;
4. Princípio da hierarquia é a organização hierárquica que está relacionada com o princípio da divisão. A organização das partes em uma estrutura hierárquica do tipo árvore sempre aumenta a comprehensibilidade.

Fonte: BARROS, 2006.

Conclusão - Unidade 3

Prezado(a) aluno(a),

Neste capítulo introduzimos o conceito de funções.

Vimos os meios para que você pudesse criar suas próprias funções e ver as vantagens de fazer isso. Entendemos que uma delas é o aproveitamento do código, que permite ao programador reutilizar sequências de código, e separar também as sequências de código, o que o torna mais legível e portável.

Este capítulo se concentrou em apresentar as principais características e vantagens das funções e apresenta o uso de funções com argumentos e como utilizar os argumentos padrões.

Nos vemos no próximo capítulo!

Material Complementar



Livro

Estruturas de Dados em C – Uma abordagem didática

Autor: SILVIO DO LAGO PEREIRA

Editora: Saraiva

Sinopse: Resultado da experiência acumulada pelo autor e direcionado a estudantes de computação e áreas correlatas, a obra apresenta os fundamentos das principais estruturas de

dados, os recursos de programação necessários para implementá-los em linguagem C e seus exemplos de aplicações práticas. Apresenta também uma extensa lista de exemplos de programas completos que podem ser diretamente executados em computador, usando o compilador Pelles C e o sistema operacional Windows. Ao final de cada capítulo, há uma série de exercícios que visam aprofundar os conceitos apresentados.

Filme

Função com retorno - Linguagem C

Ano: 2019

Sinopse: Veja como utilizar função com retorno em um programa feito em linguagem C.

Referências

ANDRÉ BACKES. **Linguagem C – completa e descomplicada.** Rio de Janeiro, Campus, 2013.

BARROS, E. A; PAMBOUKIAN, S.; ZAMBONI, L. C. **ENSINANDO PROGRAMAÇÃO ATRAVÉS DE FUNÇÕES.** WCCSETE-WORLD CONGRESS ON COMPUTER SCIENCE, ENGINEERING AND TECHNOLOGY EDUCATION.



| Unidade 4

Tipos abstratos de dados



AUTORIA

Ricardo Bittencourt Socreppa

Introdução

Prezados alunos(as)!

Neste capítulo será introduzido o conceito de listas, que podem ser usadas para colocar valores dentro de uma sequência.

Uma lista, por exemplo, tem o conceito semelhante às listas reais, aquelas que são usadas no dia a dia, com elementos e uma sequência entre esses elementos. Listas em C são extremamente importantes e permitem a criação de programa mais complexos.

Este capítulo se concentra em exaurir o assunto, apresentando as principais características e métodos das listas.

Apresenta-se o uso de listas como dois tipos específicos de estruturas de dados, a pilha e a fila. E finalmente apresenta-se as compreensões de listas, que é uma forma sucinta e fácil de criar listas.

Venha comigo!

•••••••••••••••



Desenvolver tipos abstratos de dados.



Entender sobre Listas, Filas e Pilhas.



Listas

```
id line set /*  
digit n  
key k  
key /*  
/* check code */  
/*  
/*  
/*  
/* set BK4 if correct code */  
/* reset */  
/* save the digit */
```



AUTORIA

Ricardo Bittencourt Socreppa

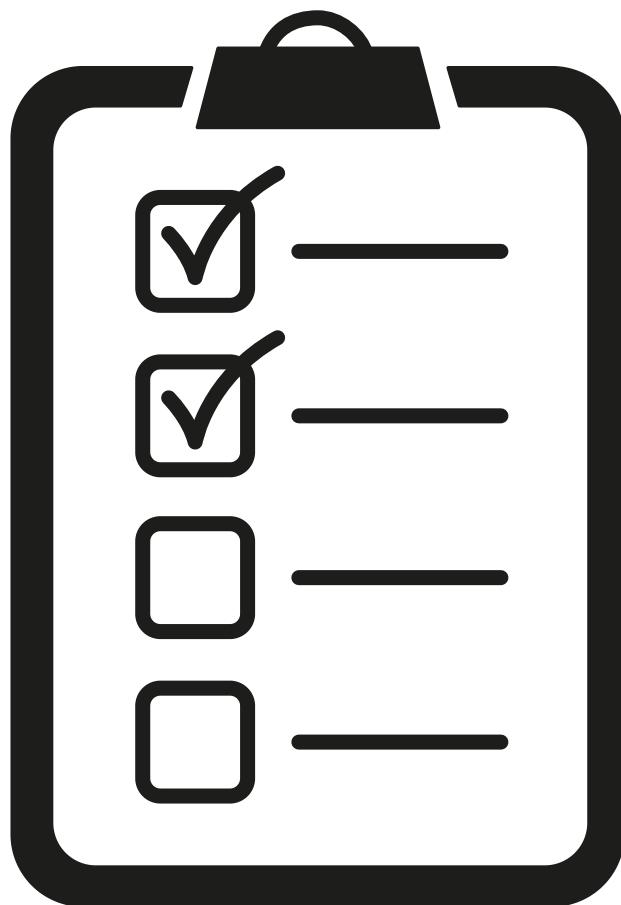
O que é uma Lista?

Conceito de lista é algo comum para as pessoas, pois trata-se de uma relação finita de itens. Todos contidos sobre um mesmo tema.

Exemplo:

- Lista de Compras;
- Lista de Casamento;
- Itens em Estoque;

Figura 1 - Exemplo de uma lista.



Fonte: elaborado pelo autor.

O que é uma Lista na Computação?

É uma estrutura de linear utilizada para armazenar e organizar dados do mesmo tipo no computador.

- Pode ser ordenada ou não;

- Pode conter elementos repetidos ou não;

Veja um exemplo na Figura 2.

Figura 2 - Exemplo de uma lista na computação



Fonte: elaborado pelo autor.

Tipos de Listas

Quanto a inserção / remoção:

- Lista convencional: Pode inserir e remover elementos em qualquer lugar da lista
- Fila: (FIFO) Insere elementos no final e remove do começo da lista
- Pilha: (LIFO) Insere e remove elementos do final da lista

Quanto a alocação de memória:

- Estática: O espaço de memória é alocado durante a compilação do programa
- É necessário definir o número máximo de elementos da lista;
- Dinâmica: o espaço de memória é alocado em tempo de execução
- A lista cresce à medida que inserimos elementos e diminui a medida que removemos;

Forma de acesso:

- Sequencial: os elementos são armazenados consecutivamente na memória
- Utiliza um Array;
- Encadeado: cada elemento pode estar em uma área distinta da memória.

Operações Básicas de uma Lista

- Criar uma Lista;
- Inserção de um elemento na lista;
- Remoção de um elemento na lista;

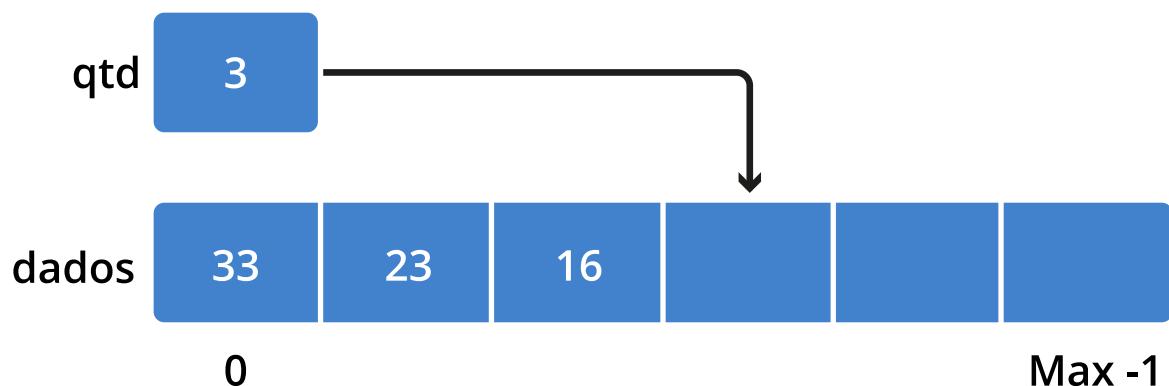
- Busca por um elemento na lista;
- Destruir uma lista;
- Tamanho de uma lista:
Cheia;
Vazia;

Listas sequenciais estáticas

É uma lista utilizando alocação estática e acesso sequencial dos elementos, e é o tipo de lista mais simples possível,

Veja a Figura 3 como utiliza array em sua implementação:

Figura 3 - Exemplo de uma lista sequencial estática



Fonte: elaborado pelo autor.

Implementação da Lista Sequencial Estática

Implementar uma lista sequencial estática que permita:

Uma lista de Alunos

- Matrícula
- Nome
- 3 Notas

Precisaremos de 03 arquivos:

- Main.c
Programa principal;

- **ListaSequencial.h**
 Protótipos das funções;
 Tipo de dado armazenado na lista;
 Tamanho do array usado na lista;
 O ponteiro “Lista”;
- **ListaSequencial.c**
 O tipo de dados “Lista”;
 Implementar as funções;

Segue os arquivos:

Código 1 - ListaSequencial.h

```
#define MAX 5
struct aluno {
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
typedef struct lista Lista;
Lista* cria_lista();
void libera_lista(Lista* li);
int insere_lista_final(Lista* li, struct aluno al);
int insere_lista_inicio(Lista* li, struct aluno al);
int insere_lista_ordenada(Lista* li, struct aluno al);
int busca_lista_pos(Lista* li, int pos, struct aluno *al);
int busca_lista_mat(Lista* li, int mat, struct aluno *al);
int remove_lista(Lista* li, int mat);
int remove_lista_inicio(Lista* li);
int remove_lista_final(Lista* li);
int tamanho_lista(Lista* li);
int lista_cheia(Lista* li);
int lista_vazia(Lista* li);
```

Fonte: elaborado pelo autor.

Código 2 - ListaSequencial.c

```
#include <stdlib.h>
#include <stdio.h>
#include "ListaSequencial.h" // Inclui os protótipos

// Definição do tipo de Lista;
struct lista {
    int qtd;
    struct aluno dados[MAX];
};
```

Fonte: elaborado pelo autor.

Criando uma lista

Vejamos a seguir um exemplo de um código mostrando como criar uma lista, observamos.

Código 3. Criando uma lista

```
// Criando uma lista
Lista* cria_lista(){
    Lista *li;
    li = (Lista*) malloc(sizeof(struct lista));
    if(li != NULL)
        li->qtd = 0;
    return li;
}
```

Fonte: elaborado pelo autor.

Destruindo uma Lista

Vejamos a seguir um exemplo de um código mostrando como destruir uma lista, observamos.

Código 4 - Destruindo uma lista

```
// Destruindo uma Lista  
void libera_lista(Lista* li){  
    free(li);  
}
```

Fonte: elaborado pelo autor.

Tamanho da Lista

Vejamos a seguir um exemplo de um código mostrando o tamanho de uma lista, observamos.

Código 5 - Tamanho de uma lista

```
// Tamanho da Lista
int tamanho_lista(Lista* li){
    if(li == NULL)
        return -1;
    else
        return li->qtd;
}
```

Fonte: elaborado pelo autor.

Lista Cheia

Vejamos a seguir um exemplo de um código mostrando quando uma lista está cheia, observamos.

Código 6 - Lista Cheia

```
// Lista Cheia
int lista_cheia(Lista* li){
    if(li == NULL)
        return -1;
    else
        return (li->qtd == MAX);
}
```

Fonte: elaborado pelo autor.

Listas Vazias

Vejamos a seguir um exemplo de um código mostrando quando uma lista está vazia, observamos.

Código 7 - Lista Vazia

```
// Lista Vazia
int lista_vazia(Lista* li){
    if(li == NULL)
        return -1;
    else
        return (li->qtd == 0);
}
```

Fonte: elaborado pelo autor.



Fila



AUTORIA

Ricardo Bittencourt Socreppa

O conceito de Fila é algo bastante comum para as pessoas como por exemplo:

- Fila de banco, cinema, mercado e etc...

Na computação a fila nada mais é do que um conjunto finito de itens (mesmo tipo) esperando por um processamento;

Exemplo:

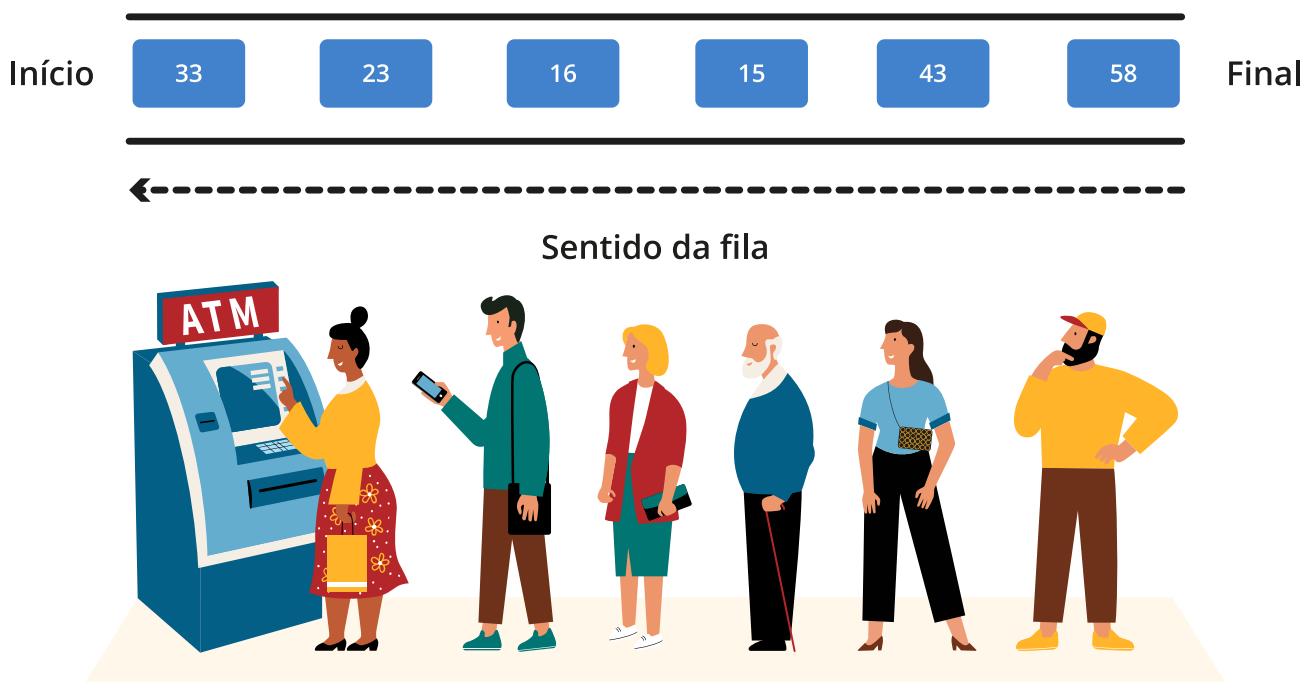
- Fila de documentos encaminhados para a impressora;

Possui uma regra de inserção e remoção:

- Insere o elemento no final da fila e remove do começo;
- Primeiro elemento que entra é o primeiro que sai (FIFO – First in First OUT).

Veja na Figura 4 a seguir um exemplo de sentido da Fila.

Figura 4 - Exemplo de sentido da Fila



Fonte: elaborada pelo autor.

Operações Básicas de uma Fila

- Criação da fila;
- Inserção de um elemento no final da fila;

- Remoção de um elemento do início da fila;
- Acesso ao elemento do início da fila;
- Destruição da fila;
- Informações
Tamanho;
Cheia;
Vazia;

Implementação da Fila Sequencial Estática

- **Precisaremos de 3 arquivos:**
Main.c
programa principal;
- FilaSequencial.h
Protótipos das funções;
Tipo de dado armazenado na fila;
O ponteiro da fila;
Tamanho do array usado na fila;
- FilaSequencial.c
O tipo de dados “Fila”
Implementar as funções;

A seguir observamos os arquivos fontes de criação de uma Fila Sequencial Estática que permita criar uma fila de pessoas com **código, nome e idade**.

Código 8 - Arquivo FilaSequencial.h

```
#define MAX 5
struct pessoa {
    int codigo;
    char nome[30];
    int idade;
};
typedef struct fila Fila;
Fila* cria_fila();
void libera_fila(Fila* fi);
int consulta_fila(Fila* fi, struct pessoa *pe);
int insere_fila(Fila* fi, struct pessoa pe);
int remove_fila(Fila* fi);
int tamanho_fila(Fila* fi);
int fila_vazia(Fila* fi);
int fila_cheia(Fila* fi);
```

Fonte: elaborado pelo autor.

Código 9 - Arquivo FilaSequencial.c

```
#include <stdlib.h>
#include <stdio.h>
#include "FilaSequencial.h"
struct fila {
    int inicio;
    int final;
    int qtd;
    struct pessoa dados[MAX];
};
```

Fonte: elaborado pelo autor.



Centro Universitário Cidade Verde

Pilha



AUTORIA

Ricardo Bittencourt Socreppa

O conceito de pilha assim como a fila é algo bastante comum para as pessoas como por exemplo:

- Pilha de roupas, tijolos, moedas e etc...

Na computação a pilha nada mais é do que um conjunto finito de itens (mesmo tipo) com a finalidade de armazenar e organizar dados;

Exemplo:

- Recursividade;

Possui uma regra de inserção e remoção:

- Os elementos são removidos na ordem inversa que são inseridos;
- Último elemento que entra é o primeiro que sai (LIFO – Last in First OUT);

Veja na Figura 5 a seguir um exemplo de sentido da Pilha.

Figura 5 - Exemplo de sentido da Pilha



Fonte: elaborada pelo autor.

Operações Básicas de uma Pilha

- Criação da pilha;
- Inserção de um elemento no topo da pilha;
- Remoção de um elemento no topo da pilha;
- Acesso ao elemento do topo da pilha;
- Destruição da pilha;
- Informações
Tamanho;
Cheia;
Vazia;

Implementação da Pilha Sequencial Estática

- **Precisaremos de 3 arquivos:**
Main.c
programa principal;
- PilhaSequencial.h
Protótipos das funções;
Tipo de dado armazenado na pilha;
O ponteiro da pilha;
Tamanho do array usado na pilha;
- PilhaSequencial.c
O tipo de dados “pilha”
Implementar as funções;

A seguir observamos os arquivos fontes de criação de uma Pilha Sequencial Estática que permita criar uma pilha de livros com **código, título e qtdepaginas**.

Código 10 - Arquivo PilhaSequencial.h

```
#define MAX 5
struct livro {
    int codigo;
    char titulo[30];
    int qtdepaginas;
};
typedef struct pilha Pilha;

Pilha* cria_pilha();
void libera_pilha(Pilha* pi);
int acessa_topo_pilha(Pilha* pi, struct livro *li);
int insere_pilha(Pilha* pi, struct livro li);
int remove_pilha(Pilha* pi);
int tamanho_pilha(Pilha* pi);
int pilha_vazia(Pilha* pi);
int pilha_cheia(Pilha* pi);
```

Fonte: elaborado pelo autor.

Código 11 - Arquivo FilaSequencial.c

```
#include <stdlib.h>
#include <stdio.h>
#include "PilhaSequencial.h" // Inclui os protótipos

// Definição do tipo de pilha;
struct pilha {
    int qtd;
    struct livro dados[MAX];
};
```

Fonte: elaborado pelo autor.



SAIBA MAIS

Engine de Inteligência Artificial

Entende-se por Inteligência Artificial (IA) para jogos, os programas que descreverão o comportamento de entidades não controladas pelo jogador, tipicamente os NPCs (Non-Player Characters).

Na maioria dos casos, o comportamento inteligente desses agentes computacionais é implementado através de máquinas de estados. Os algoritmos de máquinas de estados procuram resolver problemas formalizando diversos possíveis estados em que um elemento pode se encontrar (no caso de uma televisão, por exemplo, poderia-se ter estes estados: Desligada, Acesa e Stand-by). A transição de um estado para outro estará atrelado a algum evento que dispare esta mudança (no exemplo anterior, ao apertar a tecla <on> a TV muda do estado de desligada para Stand-by).

Desta maneira, a inteligência de um personagem de um jogo pode ser descrita por diversos estados em que o mesmo pode se encontrar (no caso de um jogo de ação, poderíamos descrever estes estados como espera, perseguição, ataque, fuga, morte, por exemplo). Este personagem deverá fazer um monitoramento constante sobre acontecimentos que possam disparar a mudança de um estado para outro. Usando o exemplo de um jogo de ação, suponha- se que o fato do jogador aproximar-se mais do que 30m do NPC faça com que o mesmo passe do estado de espera para o estado de perseguição. Entretanto é perceptível que ocorreram grandes inovações do ponto de vista gráfico, mas aspectos inteligentes das entidades computacionais não foram sofisticados com a mesma velocidade. Muitas vezes um jogo sofisticado graficamente apresenta uma IA mediana. Isto ocorre principalmente pelo fato do custo computacional que os algoritmos inteligentes e de processamento gráfico possuem. Conforme Sewald [SEW 02], técnicas de IA tais como Redes Neuronais Artificiais e Algoritmos Genéticos são pouco utilizadas em games. [...]

Fonte: CLUA; BITTENCOURT, 2005.



REFLITA

1. Pilhas e listas são chamadas de estruturas de dados, pois são estruturas que auxiliam os programadores a organizarem os dados de seus programas. A pilha é chamada de uma estrutura do tipo lifo (*last-in first-out*).
2. As estruturas de dados podem ser organizadas de formas diferentes e especiais, de acordo com a maneira com que os dados são inseridos e retirados. Um tipo particular de estruturas de dados, é a FILA.

Fonte: o autor.



Conclusão - Unidade 4

Prezado(a) aluno(a),

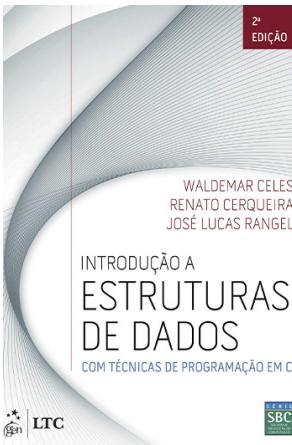
Neste capítulo foi introduzido o conceito de listas, que podem ser usadas para colocar valores dentro de uma sequência.

Vimos que uma lista, por exemplo, tem o conceito semelhante às listas reais, aquelas que são usadas no dia a dia, com elementos e uma sequência entre esses elementos. Entendemos que Listas em C são extremamente importantes e permitem a criação de programas mais complexos.

Apresentamos as principais características e métodos das listas.

Por fim, vimos que uso de listas como dois tipos específicos de estruturas de dados, a pilha e a fila. E depois apresentou-se as comprehensões de listas, que é uma forma sucinta e fácil de criar listas.

Até a próxima!



Livro

Introdução a estruturas de dados: com Técnicas de Programação em C

Autor: Waldemar Celes

Editora: Elsevier

Sinopse: A primeira edição do livro introduz programação em C e apresenta as seguintes estruturas de dados: vetor, matriz, lista encadeada, pilha, fila, árvore binária, árvore binária de busca, árvore com número variável de filhos e tabela de dispersão (hash). A 2^a edição propõe-se a ideia de estender a obra, apresentando estruturas de dados mais avançadas, a saber: vetor dinâmico, vetor de bits, lista de prioridade (heap), árvore AVL, árvore B, união-busca e grafo.

Filme

Programação em C/C++ - Pilha dinâmica

Ano: 2014

Sinopse: desenvolver os métodos de manipulação de uma das principais estruturas de dados: a pilha dinâmica!

Referências

ANDRÉ BACKES. **Linguagem C – completa e descomplicada.** Rio de Janeiro, Campus, 2013.

CLUA, ESTEBAN WALTER GONZALEZ, BITTENCOURT, JOÃO RICARDO. **DESENVOLVIMENTO DE JOGOS 3D: CONCEPÇÃO, DESIGN E PROGRAMAÇÃO.** XXIV JORNADAS DE ATUALIZAÇÃO EM INFORMÁTICA (JAI) PART OF XXIV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. 2005.



Considerações Finais

Caro(a) aluno(a),

Neste material compreendemos os princípios básicos da lógica de programação, bem como os principais comandos necessários ao desenvolvimento de algoritmos em uma linguagem de programação, para que o mesmo possa estruturar e desenvolver a resolução de problemas computacionais.

Entendemos a importância de formar um profissional ético, competente e comprometido com a sociedade em que vive, ou seja, com o desenvolvimento de perspectivas críticas, integradoras, e que construam sínteses contextualizadas.

Conhecemos e utilizamos os fundamentos, os métodos e as técnicas da lógica de programação. Compreendemos e empregamos as estruturas de dados complexas.

Por fim, ampliamos o domínio de conhecimento de algoritmos. Elaboramos códigos otimizados e avaliamos a utilização de estruturas de dados no contexto empresarial.

Continue firme nos seus estudos, pratique com dedicação os exercícios e espero lhe encontrar numa outra disciplina!!

Um forte abraço!