Comunicando-se com o Computador

Lógica de Programação Utilizando a Linguagem C

Comunicando-se com o Computador

Prof. Dr. Eduardo Manuel de Freitas Jorge
Prof. Dr. Hugo Saba Pereira Cardoso
Prof. Peterson Albuquerque Lobato
Prof. MSc. Uedson Santos Reis
Prof. MSc. Marcio Luis Valença Araújo

Lógica de Programação Utilizando a Linguagem C

Comunicando-se com o computador

Hugo Saba
Eduardo Manuel de Freitas Jorge
Claudio Reynaldo B. de Souza
(Organizadores)

Capa: Marlon Xavier e France Arnaut

Revisão: Claudio Reynaldo

Ficha catalográfica elaborada pela
Biblioteca Prof. Raul Varela Seixas /IFBA – Campus de Salvador.

C741 Comunicando-se com o computador: lógica de programação utilizando a Linguagem/Eduardo Manuel de Freitas Jorge... [et. al.]. – Salvador: EDUNEB, 2015.

67 p.

ISBN 978-85-67562-12-4

1. Lógica de programação. 2. Algoritmos. 3. Ciência da Computação. I. Jorge, Eduardo Manuel de Freitas. II. Cardoso, Hugo Saba Pereira. III. Lobato, Peterson Albuquerque. IV. Reis, Uedson Santos. V. Título.

CDU 004.42

Sumário

ÍNDICE DE ILUSTRAÇÕES

INTRODUÇÃO

Será apresentado inicialmente conhecimentos com a finalidade de capacitar o estudante sobre conceitos básicos de processamento de dados e lógica de programação, proporcionando uma maior interação com o computador.

Na interação com o computador e importante conhecer os elementos que fazem parte desse universo.

Um sistema computacional é composto de hardware, software e pessoas. *Hardware* consiste na parte física do computador, como teclado, monitor, placas e processadores. Já *Softwares* são programas ou instruções desenvolvidas com a finalidade de por em funcionamento os componentes físicos, permitindo a execução de funções desejadas pelas pessoas, que são os usuários do sistema computacional.

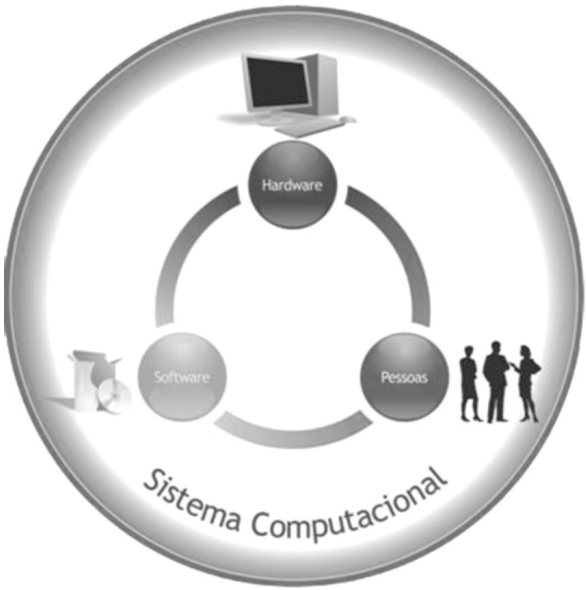

Figura 1- Sistema Computacional.
Fonte: Autor

Uma linguagem de programação é uma técnica para definir instruções que serão executadas pelos computadores. Através dessas instruções um computador pode realizar determinadas atividades no lugar dos seres humanos. Quando isso ocorre dizemos que essa tarefa ou processo foi automatizado.

O *hardware* de um Sistema Computacional é composto basicamente de 5 (cinco) elementos (ver figura1). A seguir detalham-se sucintamente estas unidades:

- **Unidades de Entrada** – periféricos que recebem as informações ou dados de entrada que serão processados pelo computador (ex.: mouse, teclado, leitor óptico, etc.).

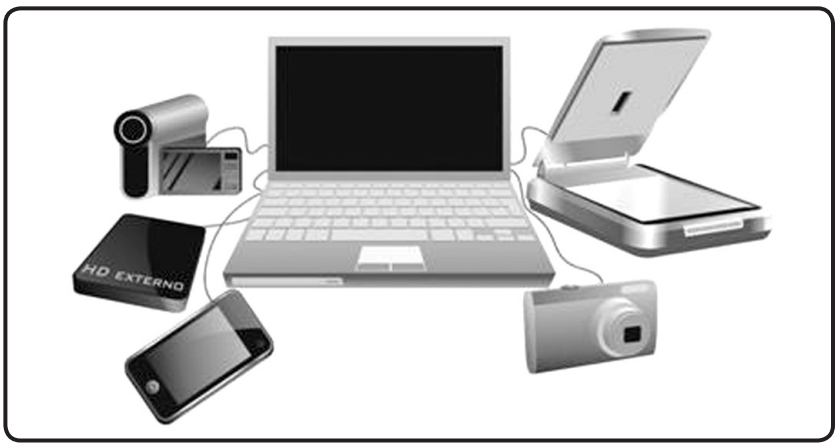

Figura 2 – Unidades de Entrada.
Fonte: Autor

- **Unidade Central de Processamento ou CPU (*Central Processing Unit*)** – onde é processada toda a informação recebida pelo computador, metaforicamente falando, podemos dizer que é o Cérebro do computador.

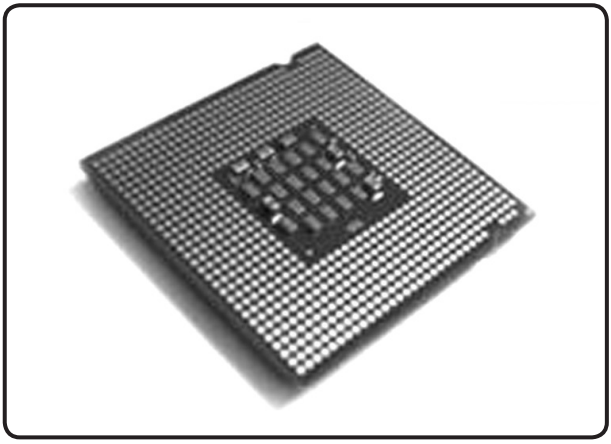

Figura 3 – Unidade Central de Processamento (UCP).
Fonte: Autor

- **Memória Principal –** é uma memória mais rápida que a secundária e normalmente utilizada para auxiliar a CPU, porém é mais cara e de uso temporário (ex.: memória RAM – *Random Access Memory*).

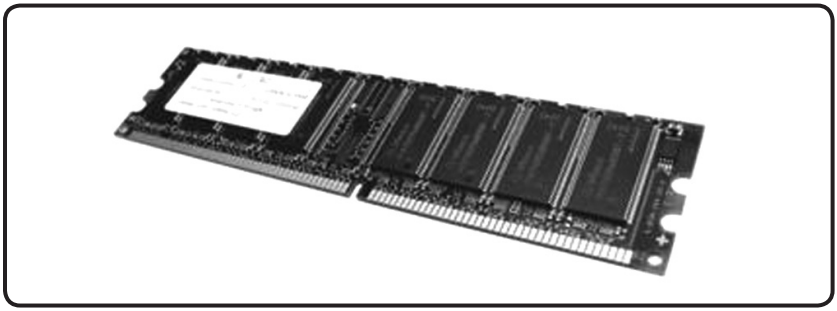

Figura 4 – Memória RAM.
Fonte: Autor

- **Memória Secundária** – é uma memória de baixo custo, e maior durabilidade para os dados. Normalmente é utiliza-

da para armazenamento de dados em grande quantidade por longos períodos de tempo. Em geral não é utilizada para auxiliar o processamento, por ser lenta para tal função (ex.: disco rígido ou HD, DVD, Pendrive etc.).

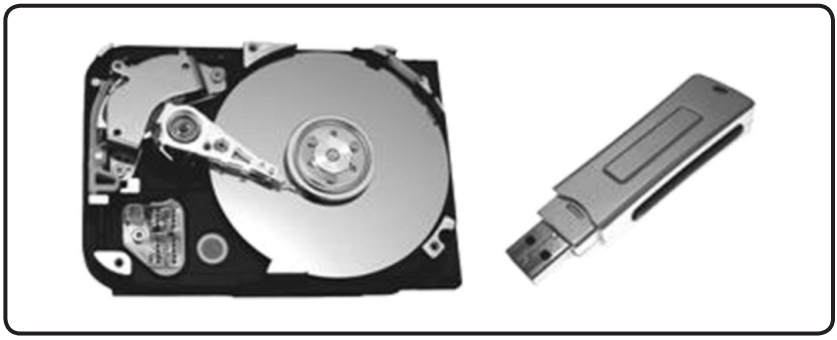

Figura 5 – Unidades Secundárias de Armazenamento.
Fonte: Autor

- **Unidades de Saída** – são periféricos que mostram ao usuário o resultado final do processamento feito pelo computador (ex.: monitor, impressora etc).

Figura 6 – Unidades de Saída.
Fonte: Autor

O *software* tem a capacidade de por um Sistema Computacional em funcionamento, proporcionando a execução de tarefas como: operadores matemáticos, impressão de gráficos, e uso de memórias secundárias etc.

O uso da **informação** de forma **automática** a partir dos sistemas computacionais conduziu o surgimento de dois novos termos:

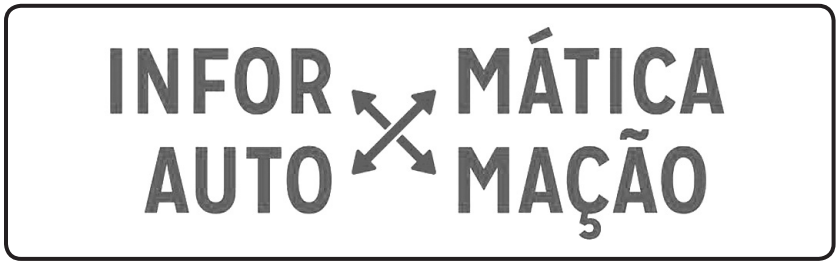

Figura 7 – Novos Termos.
Fonte: Autor.

Existe uma diversidade de linguagens de programação para a construção de software, tais como: Cobol, Pascal, Java, Objective-C, etc. Neste módulo adotaremos a linguagem *C* em nossas aplicações.

ALGORITMO

Na área da informática, o algoritmo é o inicio da criação do software, ou seja, antes de se fazer um software na linguagem de programação desejada (Pascal, C, Delphi, etc.) deve-se fazer o algoritmo desse *software*.

Cada Linguagem de Programação tem um conjunto de instruções, comandos, chamado de sintaxe, que deve ser seguida corretamente para que o programa funcione.

Formas de representação de algoritmos

- **Linguagem Natural**: Através de uma língua (português, inglês, etc.): forma utilizada nos manuais de instruções, nas receitas culinárias, bulas de medicamentos, etc.

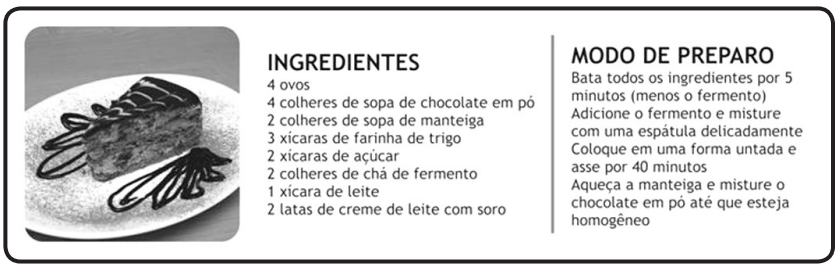

Figura 8 – Receita em Linguagem Natural.
Fonte: Autor

- **Linguagem de programação**: Conjunto de sintaxe escolhido para estruturar em forma de algoritmo.

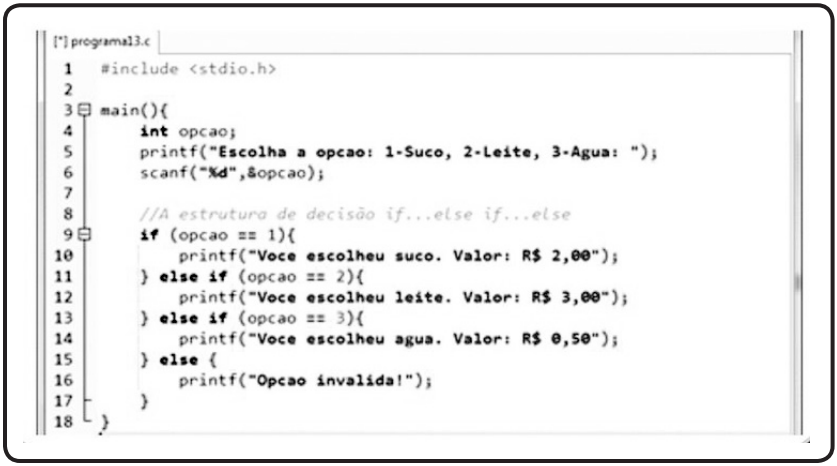

Figura 9 – Código em C.
Fonte: Autor

- **Linguagem estruturada (Portugol)** – Forma de escrever algoritmos que muito se assemelha a forma na qual os programas são escritos nas linguagens de programação (Pascal, C, Java, etc.).

```
inicio
  inteiro idade
  ler idade
  se idade < 18 entao
    escrever "de menor"
  senao
    escrever "de maior"
  fimse
fim
```

Figura 10 – Codificação em Portugol.
Fonte: Autor.

- **Fluxograma**: Através de representações gráficas.

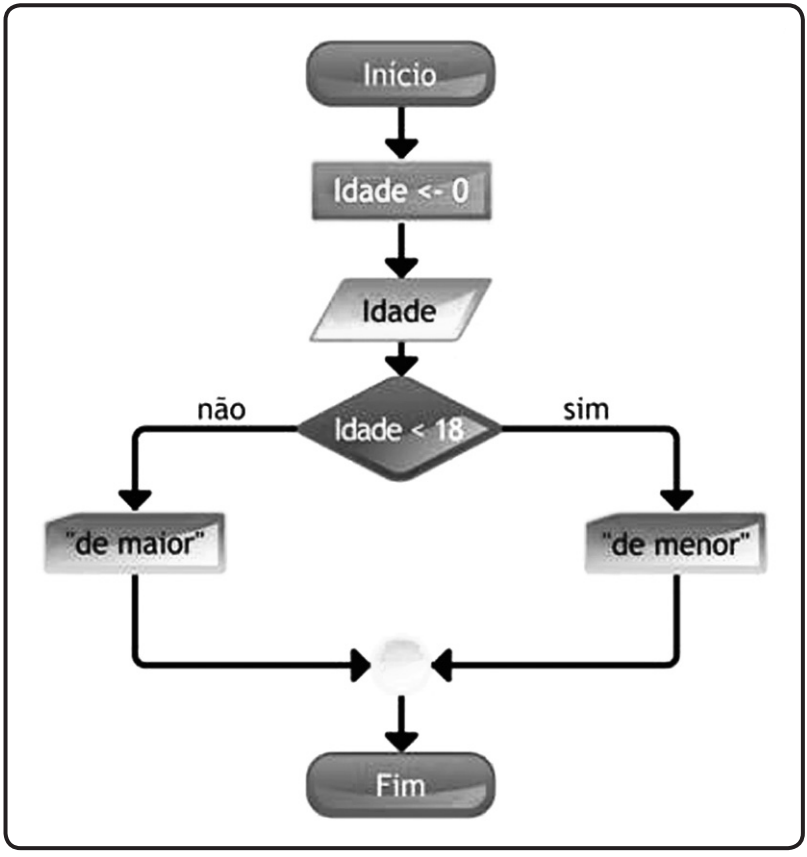

Figura 11 – Fluxograma.
Fonte: Autor.

Ano do código

Para entendermos melhor o uso de algoritmos iremos utilizar como exemplo o site denominado Ano do Código(**www.anodocodigo.org.br)**

O Ano do Código é uma iniciativa que ajuda aos interessados em programação a quebrar a primeira barreira do aprendizado.

De maneira prazerosa, como um jogo, o estudante começa a entender como um programa funciona. Através de blocos de montar, possibilita a construção de algoritmos para resolver os problemas apresentados. Aprendendo através desses algoritmos, conceitos importantes, como comandos e estruturas de programação.

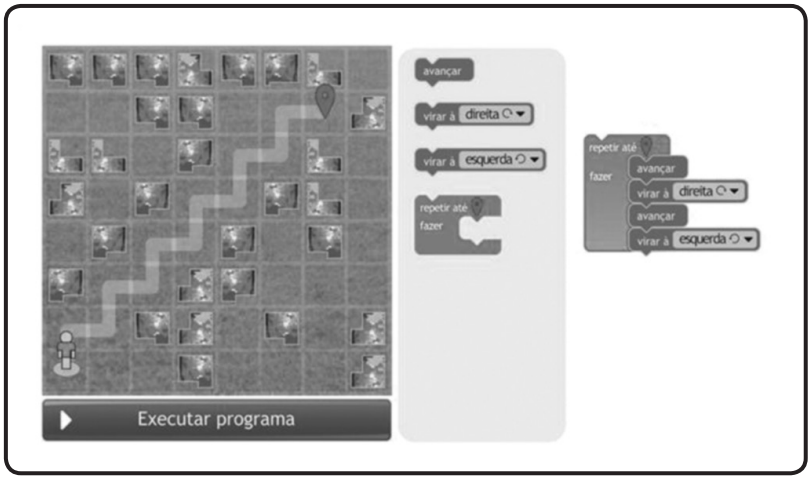

Figura 12 – Imagem do Código Gerado.
Fonte: Site AnoCódigo.

LINGUAGEM DE PROGRAMAÇÃO

A linguagem que o computador entende é chamada de linguagem de máquina, formada apenas por combinações de zeros e uns, resultando em um conjunto de códigos binários que executam comandos. Esse código é de difícil compreensão para nós.

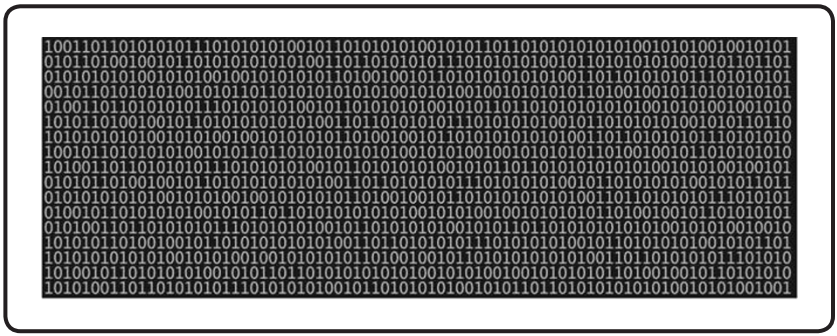

Figura 13 – Código Binário.
Fonte: Autor.

As linguagens de programação surgiram para facilitar a criação de programas, a partir do uso de sintaxe os códigos são desenvolvidos, melhorando sua compreensão para os programadores. Através do uso das linguagens de programação é possível criar comandos para serem executados por um computador.

Para que o computador entenda o que foi escrito em uma linguagem de programação é necessário à conversão para a linguagem de máquina, e isso é feito por um compilador. O Compilador é um programa que entende as duas linguagens, e transforma o programa escrito em linguagem compreensível para linguagem de máquina. A seguir iremos detalhar os principais paradigmas de programação.

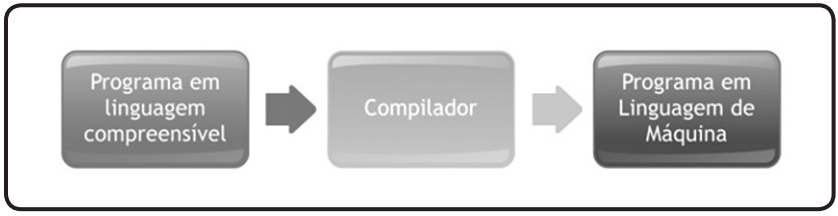

Figura 14 – Processo de Compilação.
Fonte: Autor.

Programação Estruturada

Na programação estruturada, o código escrito, utiliza de estruturas básicas: sequência, decisão e repetição, utilizando sub-rotinas chamadas funções para manter uma estrutura simples.

Programação Orientada a Objetos

A programação orientada a objetos mantém as estruturas de comando da Programação Estruturada, porém se diferencia, pois tenta simular o mundo real dentro do computador. Através de Objetos que tem propriedades (atributos) e ações (métodos), simula as características e ações que vemos em objetos no nosso mundo.

A linguagem de programação java, por exemplo, é uma das linguagens que utiliza o paradigma de orientação a objetos.

Principais Linguagens de programação

- **C**

A linguagem C é uma linguagem de programação estruturada, criada em 1972, muito popular.

- **Java**

Criada em 1990, a linguagem Java é orientada a objetos. Roda em diversas plataformas e em diversos dispositivos. Muito utilizada para o desenvolvimento de aplicativos para dispositivos móveis. que utilizam o Sistema Operacional Android.

- ***Objective*-C**

A linguagem de programação *Objective*-C, criada em 1980 é orientada a objetos, utilizada atualmente, no desenvolvimento de aplicativos para Mac OS, principalmente em dispositivos móveis da Apple.

LINGUAGEM C

A linguagem C é bastante utilizada no desenvolvimento de sistemas operacionais, mas também pode ser utilizada para o desenvolvimento de outros tipos de aplicações, como compiladores, editores de textos ou *drivers* de comunicação com *hardwares*.

DEV C++

O objetivo deste capítulo é preparar você para a instalação e utilização do DEV-C++, que é a IDE (*Integrated Development Environment*), ambiente de desenvolvimento integrado, que será utilizado para criar os programas em C.

- **Instalação**

Instalação do DEV C++

Digite no Google: **DEV C++ portable** e escolha a opção do Baixaki.

Figura 15 – Página para baixar o DEV C++.
Fonte: Site Google

Procure o link para baixar sem o instalador do baixaki.

Figura 16 – Link para baixar o DEV C++.
Fonte: Site Baixaki.

Aguarde que o arquivo vai ser baixado.

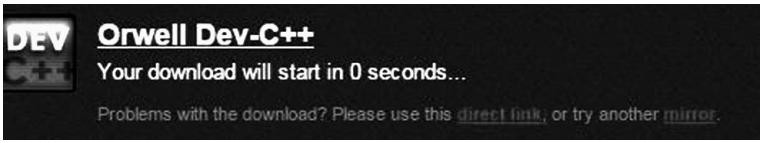

Figura 17 – Mensagem do site do DEV C++.
Fonte: Site Baixaki.

Assim que baixar, descompacte-o. Crie um atalho do aplicativo **devcpp**.

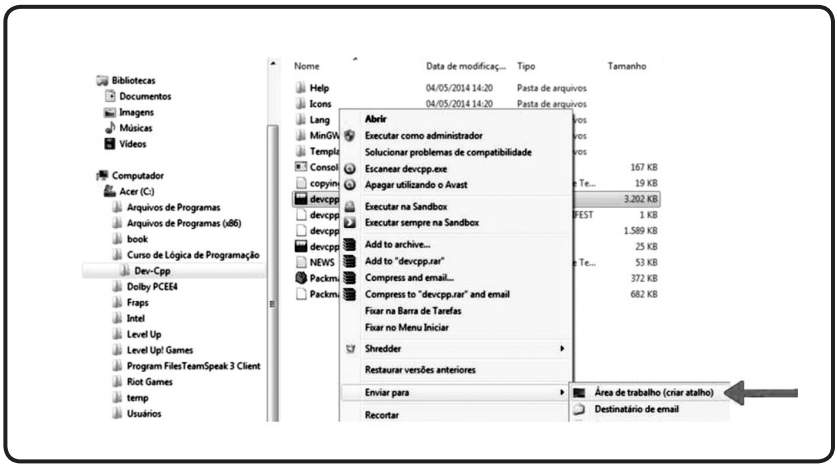

Figura 18 – Atalho DEV C++.
Fonte: Autor.

Execute o **DEV C++**. Siga os passos de Instalação.

Escolha o idioma **Português (Brasil)** e clique em **Next**.

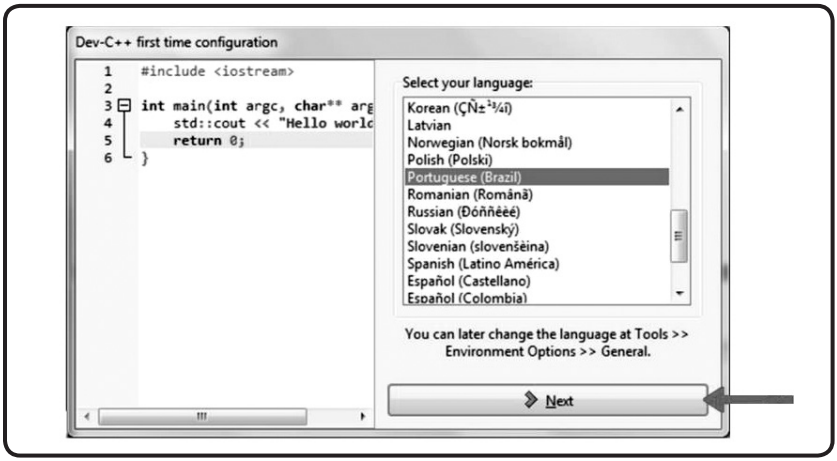

Figura 19 – Instalação do DEV C++.
Fonte: Autor.

Continue clicando em Next até o final da instalação.

Agora você está pronto para começar a escrever seus programas em C.

- **Visão Geral (Ambiente)**

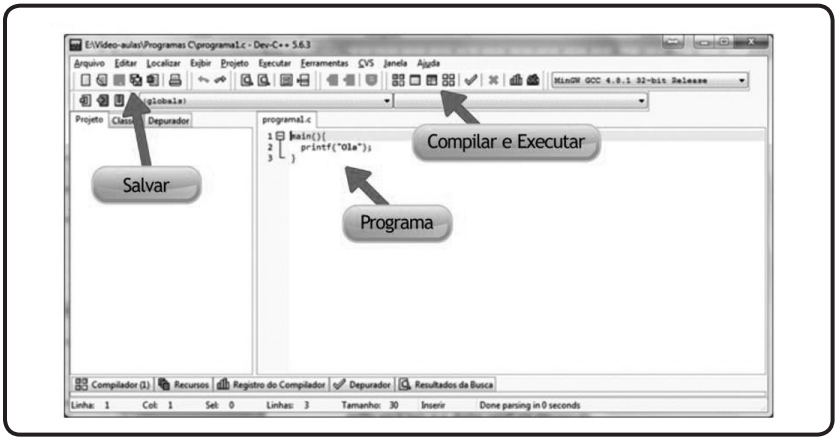

Figura 20 – Ambiente DEV C++.
Fonte: Autor.

Basicamente você deve digitar o programa na *área à direita*, salva-lo clicando no botão **salvar** (salvar como tipo C) e clicar no botão **compilar e executar** para ver o programa em execução.

Função *printf*

Usada para imprimir na tela uma cadeia de caracteres *(String)*.

Não esqueça que o C é *case sensitive* (diferencia as letras maiúsculas e minúsculas), então você tem que digitar *printf* em minúsculo, caso utilize alguma letra maiúscula na escrita do comando, o mesmo não funcionará..

Primeiro programa:

Digite o programa como mostrado abaixo.

Figura 21 – Primeiro Código C. Fonte: Autor.

O **#include <stdio.h>,** inclui a biblioteca padrão do C que contém rotinas de operações comuns necessárias para este início.

O **main**(){ **...** }, é a função principal, ela sempre deve existir, dentro dela você deve digitar o seu programa.

E o **printf("Ola!")**, imprime na tela a mensagem Ola!

Salve o programa, mudando o tipo para **C Source files (*.c)**, como mostrado na Figura 22.

Figura 22 – Salvando o Código em C.
Fonte: Autor.

E clique em **Executar** ou a tecla **F10** .

Figura 23 – Executando o Código C.
Fonte: Autor.

Figura 24 – Resultado da Execução do Código C.
Fonte: Autor.

Observação: *Todos os exemplos que traremos neste módulo, os textos digitados em C, deverão estar sem acentos*.

- **Caracteres Especiais**

Você pode utilizar usar dentro de uma cadeia de caracteres (**String**), caracteres especiais para fazer funções diferentes.

Caractere	Função
\n	Pula Linha
\a	Sinal Sonoro
\t	Tabulação
\’	Aspas simples
\”	Aspas duplas
\?	Interrogação (?)
\\	A própria \

- **Inserir linha dentro do printf**.

Este programa mostra o uso do caractere especial \n, que permite pular linha.

programa2.c

```
1  #include <stdio.h>
2
3  main(){
4      printf("Curso de C \nCurso \nLogica de Programacao");
5  }
6
7
```

Figura 25 – Uso do Caractere Especial \n.
Fonte: Autor.

```
Curso de C
Curso
Logica de Programacao
--------------------------------
```

Figura 26 – Resultado da Execução com o \n.
Fonte: Autor.

- **Emitir sinal sonoro**.

O caractere **\a** emite um sinal sonoro.

programa3.c

```
#include <stdio.h>

main(){
    printf("\a");
}
```

Figura 27 – Uso do Caractere Especial \a.
Fonte: Autor.

Fazer um programa que imprima na tela:

1 – Cadastro de Clientes
2 – Cadastro de Fornecedores
3 – Cadastro de Produtos
4 – Vendas
0 – Fim do Programa

programa4.c

```
#include <stdio.h>

main(){
    printf("1\t-Cadastro de Clientes\n");
    printf("2\t-Cadastro de Fornecedores\n");
    printf("3\t-Cadastro de Produtos\n");
    printf("4\t-Vendas\n\n");
    printf("0\t-Fim do Programa");
}
```

Figura 28 – Programa para Imprimir na Tela Informações.
Fonte: Autor.

Comentários

À medida que seu programa for aumentando, você sentirá a necessidade de explicar o que o trecho de código faz. Para isso existe a possibilidade de escrever comentários.

Utilize **//** para comentar uma linha.

Figura 29 – Usando Barras para Comentários.
Fonte: Autor.

Utilize **/* */** para comentar várias linhas

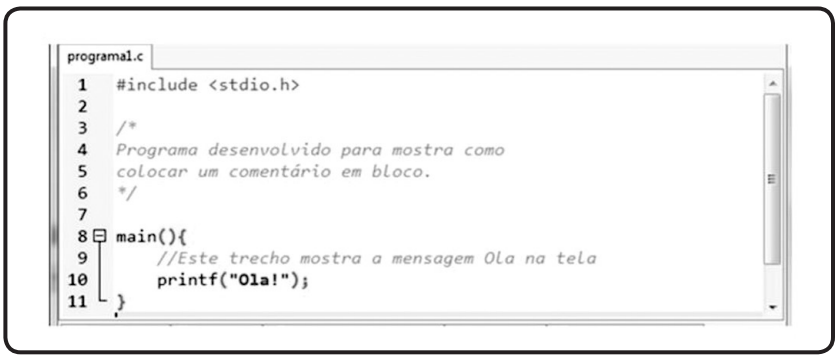

Figura 30 – Usando Barras e Asterisco para Comentários.
Fonte: Autor.

Variáveis

Um dos primeiros conceitos para construção de um algoritmo é o conceito de variável. Sinteticamente podemos definir como um espaço na memória que pode armazenar alguma informação.

Exemplos:

Nome: Maria

Idade: 12

Endereço: Rua das flores

CEP: 42.456-902

CPF: 345.345.112-02

Altura: 1,67

Nascimento: 12/07/2003

São variáveis: nome, idade, endereço, CEP, CPF, altura e nascimento.

A variável "nome" contém a informação "Maria", a variável "Idade" tem a informação "12", e assim todas as demais variáveis contendo os seus respectivos valores.

Cada variável deve ser definida informando o nome e o tipo da informação que ela armazenará, como mostrado na tabela abaixo.

Variável	Informação	Tipo
Nome	Maria	texto
Idade	12	inteiro
endereço	Rua das Flores	texto
cep	42456-902	texto
cpf	345.345.112-02	texto
altura	2,67	decimal
nascimento	12/07/2003	data

Na linguagem C existem cinco tipos básicos de dados que podem ser associados a variáveis, são eles:

Tipo	Descrição
int	números inteiros (-32768 até 32767);
float	números reais, com casas decimais (1.1e-38 até 3.4e+38);
char	para representar um caractere (-128 até 127);

Por exemplo, se queremos somar dois números inteiros poderíamos definir três variáveis int **x,y,z**; depois indicar que **x**=10 e **y**=20. Por fim, efetuar a operação e utilizar a variável **z** para receber o resultado da operação, logo **z** = **x** + **y.**

Figura 31 – Uso de Variáveis.
Fonte: Autor.

Constantes

Outro conceito associado às variáveis são as constantes. Quando definimos um uma variável como constante significa que a mesma não pode sofrer modificação no seu valor.

Por exemplo, const float pi = 3.1415926535, logo a constante pi nunca poderá mudar o seu valor.

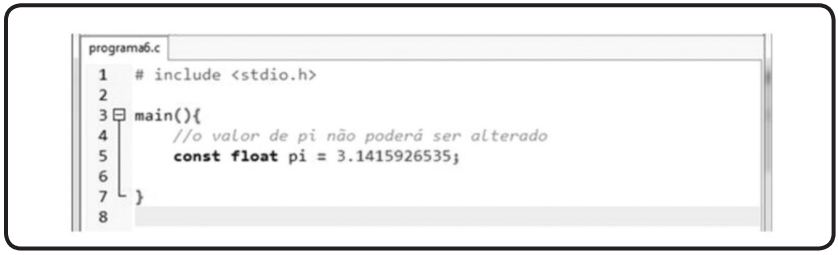

Figura 32 – Uso de Constantes.
Fonte: Autor.

Observação: const é um comando na linguagem C que determina que a variável seja constante, ou seja, seu valor será sempre o mesmo (definido no momento de sua declaração).

Imprimindo variáveis com o printf

Para imprimir você deve utilizar o formatador correto para o tipo de dados da variável ou constante.

Por exemplo, para o tipo de dados **int** utiliza-se o formatador **%d**. Observe o exemplo abaixo que ele ilustrará melhor essa questão:

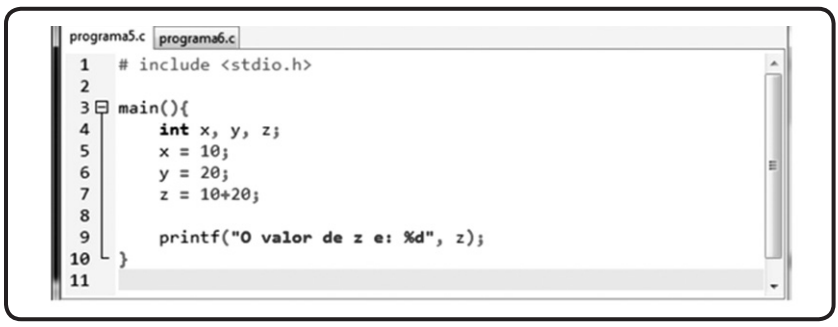

Figura 33 – Uso do Printf.
Fonte: Autor.

Figura 34 – Resultado do Uso do Printf.
Fonte: Autor.

Outra questão importante é como formatar a apresentação de números com casas decimais. O exemplo abaixo mostra como imprimir um número decimal usando o **printf** e o formatador **%f**.

programa6.c

```
1  # include <stdio.h>
2
3  main(){
4      //o valor de pi não poderá ser alterado
5      const float pi = 3.141592;
6      printf("O valor de pi e: %f",pi);
7  }
8
```

Figura 35 – Uso do Formatador %f.
Fonte: Autor.

```
O valor de pi e: 3.141592
```

Figura 36 – Resultado do Uso do Formatador %f.
Fonte: Autor.

Se você quer controlar o número de casas decimais, basta acrescenta a precisão. No exemplo anterior, para mostrar o valor de pi com apenas duas casas decimais é necessário mudar o formatador para **%.2f**, como observado na imagem abaixo.

programa6.c

```
1  # include <stdio.h>
2
3  main(){
4      //o valor de pi não poderá ser alterado
5      const float pi = 3.141592;
6      printf("O valor de pi e: %.2f",pi);
7  }
8
```

Figura 37 – Uso do Formatador %.2f.
Fonte: Autor.

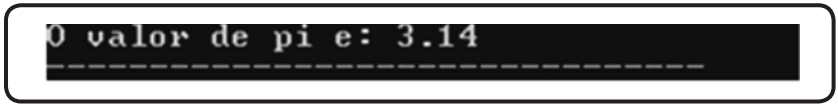

Figura 38 – Resultado do Formatador %.2f.
Fonte: Autor

A seguir, listam-se os formatadores e seus respectivos tipos de dados:

- **%d** – exibe um inteiro decimal
- **%i** – exibe um inteiro decimal com sinal
- **%f** – exibe valores de ponto flutuante
- **%e** – exibe valores de ponto flutuante em notação exponencial
- **%c** – exibe caracteres
- **%s** – exibe strings

Função scanf

Já a função **scanf()** serve para obter dados do teclado. Os dados que vão ser obtidos serão armazenados em variáveis. Os formatadores que você aprendeu na função **printf()** serão agora utilizados para fazer uma correlação com o tipo de dado que será digitado e a variável que irá armazenar o dado.

No exemplo seguinte, o programa pede ao usuário para digitar a sua idade. O valor digitado, que deverá ser um número inteiro, como mostra o formatador "%d", será gravado na variável idade, declarada como inteiro também.

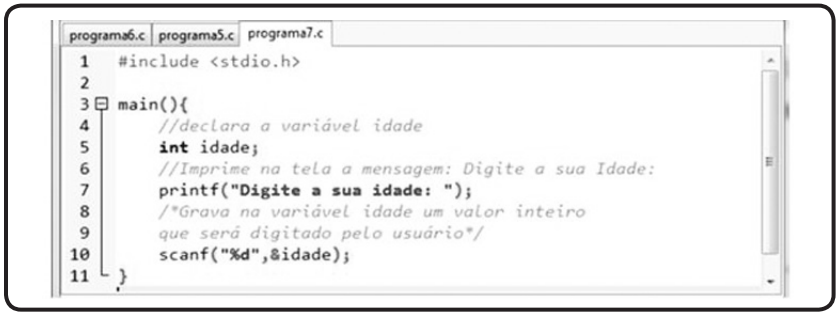

Figura 39 – Uso do Comando Scanf.
Fonte: Autor

O próximo exemplo demonstra um programa que solicita um número ao usuário, o multiplica por 3 e exibe o resultado.

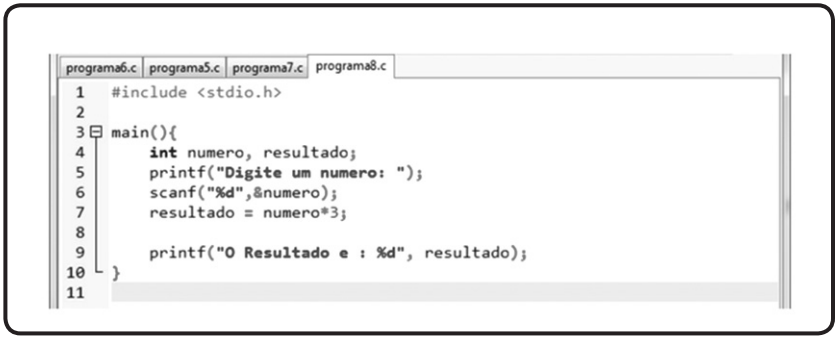

Figura 40 – Código Exemplo com Multiplicação.
Fonte: Autor.

Figura 41 – Resultado do Código Exemplo de Multiplicação.
Fonte: Autor.

Operadores Matemáticos

Os operadores matemáticos são:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão)

Abaixo, um exemplo usando os operadores matemáticos:

programa17.cpp programa18.c

```
1  #include <stdio.h>
2
3  main(){
4      int numero1, numero2;
5      numero1 = 20;
6      numero2 = 10;
7
8      printf("Soma: %d \n",numero1+numero2);
9      printf("Subtracao: %d \n",numero1-numero2);
10     printf("Multiplicacao: %d \n",numero1*numero2);
11     printf("Divisao: %d \n",numero1/numero2);
12 }
```

Figura 42 – Uso de Operadores Matemáticos.
Fonte: Autor.

Operadores Relacionais

Os operadores relacionais são usados quando precisamos comparar duas informações. Usados principalmente com as estruturas *if*, *while* e *for*, que iremos estudar posteriormente. São eles:

Operador	Descrição
==	Igual
!=	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual

A seguir um exemplo usando os operadores relacionais. Note que o programa da a resposta 0 para falso e 1 para verdadeiro.

programa17.cpp programa19.c

```
#include <stdio.h>

main(){
    int numero1, numero2;
    numero1 = 20;
    numero2 = 10;
    printf("Numero1 = %d, Numero2 = %d \n", numero1, numero2);
    printf("Igual: %d \n",numero1==numero2);
    printf("Diferente: %d \n",numero1!=numero2);

    printf("Maior: %d \n",numero1>numero2);
    printf("Menor: %d \n",numero1<numero2);

    printf("Maior ou Igual: %d \n",numero1>=numero2);
    printf("Menor ou Igual: %d \n",numero1<=numero2);
}
```

Figura 43 – Uso de Operadores Relacionais.
Fonte: Autor.

É importante distinguir (=) de (==). O primeiro atribui um valor e o segundo comprara expressões.

Operadores Lógicos

Geralmente usando junto com os operadores relacionais nas estruturas *if*, *while* e *for.* São eles:

Operador	Descrição
&&	AND (e)
||	OR (ou)
!	NOT (não)

No próximo exemplo, o usuário digita um número e o programa diz se ele está entre 0 e 10, respondendo 0 se falso e 1 se verdadeiro.

programa17.cpp programa20.c

```
1  #include <stdio.h>
2
3  main(){
4      int numero;
5
6      printf("Digite um numero: ");
7      scanf("%d",&numero);
8
9      printf("O numero esta entre 0 e 10? %d", (numero>0) && (numero <10));
10 }
11
```

Figura 44 – Uso de Operadores Lógicos.
Fonte: Autor.

Manipulando *Strings*

String é uma matriz (veremos o conceito de matrizes mais na frente) que armazena textos. As *strings* em C exigem um cuidado especial, pois trabalha caractere à caractere, posição por posição.

No exemplo abaixo, demonstra como declarar uma variável para receber uma *string*.

programa9.c

```
1  #include <stdio.h>
2
3  main(){
4      char nome[30] = "Logica de Programacao";
5
6      printf("%s", nome);
7  }
8
```

Figura 45 – Manipulando Strings.
Fonte: Autor.

Observe que foi criada uma variável nome, tipo char, tamanho 30, com o dado "Lógica de Programação". Para imprimir a string na tela através do printf, foi necessário o uso do formatador "%s".

Obs: Para utilizar as funções de tratamento de strings é necessário a inclusão de mais uma biblioteca, a string.h.

Abaixo estão as funções mais importantes para o tratamento de strings.

Função	Descrição
gets(var1)	Entrada de dados pelo teclado
strcpy(var1, var2)	Copia var2 para var1
strcat(var1, var2)	Concatena var2 no final de var1
strlen(var1)	Retorna o tamanho de var1
strcmp(var1, var2)	Compara var1e var2. Se var1 == var2 retorna 0. Se var1 < var2 retorna valor < 0. Se var1> var2 retorna valor > 0.

O exemplo seguinte pede ao usuário que digite seu nome.

Figura 46 – Utilizando o Get.
Fonte: Autor.

Foi declarado a variável nome, tipo *char*, tamanho 30, imprimiu na tela a mensagem "Digite seu nome:" e através da função *gets*, gravou a *string* digitada pelo usuário na variável nome.

O exemplo seguinte pede para o usuário digitar seu nome e sobrenome e concatena os dois através da função **strcat**.

programa9.c programa10.c

```
1   #include <stdio.h>
2   #include <string.h>
3
4   main(){
5       char nome[30], sobrenome[30];
6       printf("Digite seu nome: ");
7       gets(nome);
8       printf("Digite seu sobrenome: ");
9       gets(sobrenome);
10
11      //Acrescenta o sobrenome ao nome
12      strcat(nome, sobrenome);
13      printf(nome);
14  }
```

Figura 47 – Utilizando o Strcat.
onte: Autor.

```
Digite seu nome: Maria
Digite seu sobrenome: Joaquina
Maria Joaquina
```

Figura 48 – Resultado do Uso do Strcat.
Fonte: Autor.

No próximo exemplo, o programa retorna a quantidade de caracteres digitados pelo usuário, através da função **strlen**.

programa9.c programa11.c

```
1   #include <stdio.h>
2   #include <string.h>
3
4   main(){
5       char nome[100];
6       int quant;
7       printf("Digite seu nome: ");
8       gets(nome);
9
10      //Grava na variável quant o tamanho da string da variável nome.
11      quant = strlen(nome);
12      printf("Tamanho: %d", quant);
13  }
14
```

Figura 49 – Uso do Strlen.
Fonte: Autor.

Estrutura de Decisão – *(if e switch)*

O comando de seleção é utilizado para determinar que um comando o um bloco de comandos serão executados com base em uma condição booleana.. Vamos a uma breve revisão da Tabela Verdade. Fique atento (a) a notação dos operadores booleanos na linguagem C. Como vimos anteriormente o ! significa negação, || significa ou (or) e && que significa e *(and)*.

Operadores clássicos da lógica booleana são reconhecidos pela linguagem de programação C e podem ser utilizados nos programas nos comandos *if*, *while*, *do while* e *for* (veremos mais adiante). A tabela abaixo apresenta os operadores not (negação), or (ou) e and (e) na notação da linguagem C. Para saber mais sobre a álgebra booleana acesse http://www.tecmundo.com.br/programacao/1527-logica-booleana-saiba-um-pouco-mais-sobre-esta-logica-e-como-ela-funciona.htm.

A B	!A	(A||B)	(A&&B)
V V	F	V	V
V F	F	V	F
F V	V	V	F
F F	V	F	F

Comando *If*

O comando *if* é um comando de seleção condicional. Na sua estrutura após a palavra if deve-se indicar uma expressão condicional, ou seja, uma expressão booleana. Por exemplo, *if* ((x==10) && (y==20)), neste caso temos três condições booleanas. Se x=10 e y=30 a expressão será formada por ((v)&&(F)). Logo o resultado após o processamento é F. Uma dica importante é sempre manter uma condição booleana entre ( ). No exemplo anterior a montagem dos parâmetros ficou assim ((..) && (..)). Observe que o parêntese mais ao estremo atua sobre a condição booleana resultante de (V)

&& (F). Outra dica importante é não confundir = como atribuição, ou seja, x=10 e == de comparação x==10. A seguir apresenta-se mais detalhes do comando *if*.

Estrutura:	Exemplo:

if (condição) {
Bloco de Comando;
} else {
Bloco de Comando;
} if else (condição){
Bloco de Comando;
}
	if (cor == VERDE) {
	printf(“AMARELO”);
	} else if (cor == AMARELO) {
	printf(“VERMELHO”);
	} else if (cor == VERMELHO)
	{
	printf(“VERDE”);
	}

Uma estrutura de decisão examina uma ou mais condições e decide quais instruções serão executadas. O comando *if* é uma estrutura de decisão. A estrutura é composta de *if, else if, else*, que traduzindo: se, senão se, senão. Para a condição testada, utilizam-se operadores de comparação e relacionais, estudados anteriormente.

No programa abaixo é um exemplo utilizando a estrutura de decisão *if*. Ele informa se a pessoa é maior ou menor de idade, dependendo da idade digitada.

programa12.c

```
#include <stdio.h>

main(){
    int idade;
    printf("Digite sua idade: ");
    scanf("%d",&idade);

    //A estrutura de decisão if...else
    if (idade < 18){
        printf("Voce e menor de idade!");
    } else {
        printf("Voce e maior de idade!");
    }
}
```

Figura 50 – Utilizando o IF.
Fonte: Autor.

No próximo exemplo, o usuário digita a opção desejada e o programa retorna a escolha selecionada.

[*] programa13.c

```
#include <stdio.h>

main(){
    int opcao;
    printf("Escolha a opcao: 1-Suco, 2-Leite, 3-Agua: ");
    scanf("%d",&opcao);

    //A estrutura de decisão if...else if...else
    if (opcao == 1){
        printf("Voce escolheu suco. Valor: R$ 2,00");
    } else if (opcao == 2){
        printf("Voce escolheu leite. Valor: R$ 3,00");
    } else if (opcao == 3){
        printf("Voce escolheu agua. Valor: R$ 0,50");
    } else {
        printf("Opcao invalida!");
    }
}
```

Figura 51 – Uso do IF com Escolha.
Fonte: Autor.

Obs: Uma dica importante é não confundir = como atribuição, ou seja, x=10 e == de comparação x==10. A seguir apresenta-se mais detalhes do comando if.

Já o **Operador Ternário** (Condição **?** verdade : falso) é uma forma de fazer um if-then-else em uma um único comando. Você conseguiria fazer tudo que o operador ternário faz com um simples *if*, porém não é incomum a escrita deste comando pelos desenvolvedores. Assim, para evitar que você se depare com este comando e não o entenda vale a pena estudá-lo.

O **Operador Ternário** (?) deve ser colocado depois da condição booleana, após ele vem o resultado caso a condição seja verdadeira seguida de " : " e o resultado caso a condição seja falsa. Sendo assim, a estrutura de um *if-then-else* utilizando o operador ternário (?) é a seguinte:

CondB ? ResV : ResF
☐ CondB (condição booleana)
☐ ResV (resultado caso verdade)
☐ ResF (resultado caso falso)
	Exemplo:
	int x = 10;
	y = (x > 9 ? 100 : 200);
	Ou seja: y = 100

Estruturas de Repetição

Comando de Repetição *While*

O **Comando de Repetição *while*** permite que você faça o que chamamos em programação de laço, ou seja, você consegue programar algo que se repita diversas vezes. Portanto, o comando *while* (**condução**) mantém uma estrutura de repetição enquanto uma condição for verdadeira. Observe a sua estrutura e o trecho do programa a seguir que utiliza o comando *while* para incrementar o valor de x até 10 (dez):

Estrutura:
while (condição) {
(Bloco de comandos)
}
	Exemplo:
	int x=0;
	while (x <= 10) {
	x++; // x = x + 1;
	}

Comando de Repetição *do-while*

O **Comando de Repetição *do-while*** tem a mesma função do while, a diferença é que o *do-while* só verifica a condição de parada após a primeira iteração, conforme estrutura abaixo.

```
do {
   (Bloco de Comandos)
} while (condição);
```

Desta forma, o bloco de comandos é executado pelo menos uma vez no *do-while*, no *while* o bloco de comandos pode não ser executado. Deve ser utilizado em códigos que necessitem fazer a checagem da condição depois de executar a primeira iteração do laço. Abaixo temos um exemplo de uso *do-while*, onde a estrutura do comando também imprime e incrementa x em 10 (dez).

```
int x = 0;
do {
   x++;
   printf("%d", x);
} while(x < 10);
```

Faça o exemplo seguinte, e observe que o programa imprime na tela os números de 1 a 10.

programa14.c

```
1  #include <stdio.h>
2
3  main(){
4      int numero = 1;
5
6      //Enquanto o numero for menor ou igual a 10 faça...
7      while (numero <= 10){
8          printf("%d ",numero);
9
10         //incremento na variável numero
11         numero = numero+1;
12     }
13 }
```

Figura 52 – Imprimir Números de 1 a 10.
Fonte: Autor.

Dica: o incremento de uma variável pode ser feito também através do ++. No exemplo anterior poderia ser numero++.

No programa abaixo, o usuário continua escolhendo uma opção até digitar o número 0.

programa15.c

```
main(){
    int opcao = 1;

    while (opcao != 0){
        printf("Escolha a opcao: 1-Suco, 2-Leite, 3-Agua: ");
        scanf("%d",&opcao);

        //A estrutura de decisão if...else if...else
        if (opcao == 1){
            printf("Voce escolheu suco. Valor: R$ 2,00\n");
        } else if (opcao == 2){
            printf("Voce escolheu leite. Valor: R$ 3,00\n");
        } else if (opcao == 3){
            printf("Voce escolheu agua. Valor: R$ 0,50\n");
        } else {
            printf("Opcao invalida!\n");
        }
    }
}
```

Figura 53 – Código para Solicitar Opção até Digitar Zero.
Fonte: Autor

- Estrutura de Repetição ***For***

O **comando de repetição *for*** é mais uma forma de fazer um laço (uma interação). Neste caso é utilizado quando se quer determinar a quantidade de iteração das repetições. No caso do *for*, a condição de parada não é uma sentença booleana simples e sim uma contagem até um determinado número, que pode ser uma variável normal ou uma constante.

O comando for é bastante flexível e com criatividade pode-se alcançar variações de soluções superinteressantes. Abaixo temos a estrutura do comando de repetição for e um exemplo básico onde novamente incrementamos uma variável x de 1 (um) até 10 (dez):

Estrutura:	Exemplo:

for (inicialização; condição; incremento) {
(Bloco de Comando)
}
	int x;
	for(x = 1; x <= 10; x++) {
	printf("%d", x);
	}

No exemplo abaixo o programa imprime na tela os números de 1 a 10 utilizando o *for*, da mesma forma que o exemplo usando o *while*.

programa16.c

```
#include <stdio.h>

main(){
    int numero;

    //Para numero = 1 até numero <= 10, incremento 1, faça
    for (numero=1; numero <= 10; numero++){
        printf("%d ",numero);
    }
}
```

Figura 54 – Utilizando o For com Incremento de 1.
Fonte: Autor.

No exemplo seguinte, usamos o decremento (--) para imprimir na tela os números na ordem inversa.

programa17.cpp

```
#include <stdio.h>

main(){
    int numero;

    //Para numero = 10 até numero >= 1, decremento 1, faça
    for (numero=10; numero >= 1; numero--){
        printf("%d ",numero);
    }
}
```

Figura 55 – Utilizando o For com Decremento de 1.
Fonte: Autor.

Matrizes

Uma Matriz é um conceito matemático muito utilizado na computação para a construção de software. Por exemplo, o software Microsoft Excel é uma grande matriz e o usuário pode trabalhar com as linhas e as colunas colocando informações que deseja armazenar e processar.

Uma matriz na linguagem de programação C é uma coleção de variáveis do mesmo tipo que podem ser referenciadas por um único nome.

- **Vetor**

Um vetor, também conhecido como matriz unidimensional tem uma estrutura que só possui um índice. Este índice na definição da variável indica tamanho do vetor.

Estrutura

```
tipo var[índice]
```

A inicialização de uma matriz depende do tipo de variáveis que ela irá armazenar. Por exemplo, se for do tipo *int* os números que irão preencher a variável deverão vir entre chaves:

```
int  num[10] = {1,2,3,4,5,6,7,8,9,10};
```

Neste exemplo a variável num é um vetor de 10 posições.

Para recuperar um elemento de um vetor, devemos informar a posição na qual este se encontra. A posição começa a contar a partir do 0. Conforme nosso exemplo, o número 1 se encontra na posição 0 e o número 5 na posição 4. Conforme a tabela abaixo:

Posição	0	1	2	3	4	5	6	7	8	9
num	1	2	3	4	5	6	7	8	9	10

Figura 56 – Vetor de 10 Posições.
Fonte: Autor.

Como demonstra a Figura 57, é declarado um vetor de 5 posições do tipo inteiro. É impresso na tela o primeiro numero representado pelo índice 0 e o último representado pelo índice 4.

programa22.c | programa21.c

```c
#include <stdio.h>

main(){
  int numero[5];

  numero[0] = 10;
  numero[1] = 20;
  numero[2] = 30;
  numero[3] = 40;
  numero[4] = 50;

  printf("O primeiro numero e : %d \n", numero[0]);
  printf("O ultimo numero e : %d \n", numero[4]);
}
```

Figura 57 – Código com Vetor de 5 Posições.
Fonte: Autor.

O exemplo seguinte utiliza a estrutura de repetição *for* para solicitar cinco números ao usuário e em seguida imprimi-los na tela. Esta estrutura é muito utilizada para trabalhar com matrizes.

programa22.c | programa21.c

```c
#include <stdio.h>

main(){
  int numero[5], i, soma=0;
  float media;

  //Solicita 5 números ao usuário
  for(i=0; i < 5; i++){
      printf("Digite o %d numero: ", i+1);
      scanf("%d",&numero[i]);
  }

  //Imprime vetor
  for (i=0; i<5; i++){
    printf("%d ", i+1, numero[i]);
    soma = soma+numero[i];
  }
  media = soma/5;

  printf("\nO primeiro numero e : %d \n", numero[0]);
  printf("O ultimo numero e : %d \n", numero[4]);
  printf("A soma e : %d \n", soma);
  printf("A media e : %.2f \n", media);

}
```

Figura 58 – Código Utilizando o *For* com Vetor.
Fonte: Autor.

Para o tipo *char*, o texto deve vir entre aspas. Este tipo de vetor já vimos anteriormente quando estudamos *strings*.

char texto[14] = “Eu gosto de C”;

- **Matrizes Bidimensionais**

As **Matrizes Bidimensionais** são como os vetores, porém dentro da matriz bidimensional não teremos uma coleção de variáveis e sim uma coleção de vetores.

Estrutura:

int numero [qtd_linhas] [qtd_colunas];

A inicialização da matriz bidimensional segue a mesma lógica do vetor, a diferença é que a quantidade total de elementos se dá pela multiplicação dos índices de linha e coluna da matriz bidimensional.

Vejamos o exemplo abaixo para entender melhor:

programa22.c | programa21.c | programa23.c

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int l,c;
6      int matriz[10][10];
7      int count = 1;
8      for(l=0; l < 10; l++)
9          for(c=0;c < 10; c++){
10             matriz[l][c]=count;
11             count++;
12         }
13     for(l=0; l < 10; l++){
14         for(c=0;c < 10; c++){
15             printf("%d ",matriz[l][c]);
16         }
17         printf("\n");
18     }
19 }
```

Figura 59 – Código Utilizando Matriz Bidimensional.
Fonte: Autor.

Funções

As **funções** permitem a aplicação do conceito de modularização. A técnica consiste em dividir um problema em partes menores, criando unidades funcionais pequenas de mais fácil compreensão e desenvolvimento.

Uma função tem como objetivo ampliar a reutilização e facilitar manutenções corretivas e evolutivas. É recomendável que as funções tenham por volta de 10 (dez) linhas de código. Quando as construímos podemos chamá-las em outras funções em qualquer ponto de um programa.

Estrutura:

Tipo nome_da_função (**parâmetros**)
{
instruções;
}

A seguir mostraremos a estrutura de uma função e detalharemos seus elementos:

- **Tipo**: determina o tipo do valor a ser retornado pela função.
- **Parâmetros**: lista de nomes de variáveis separadas por vírgulas e seus tipos associados: quando a função é usada os parâmetros recebem valores.
- **Instruções**: local onde se programa o código da função.

A seguir temos um exemplo de declaração de uma função e sua chamada:

programa22.c | programa21.c | programa23.c | programa24.c

```c
#include <stdio.h>

//Declaração da função
int soma(int x, int y);

int main() {
    int num1, num2;
    printf("Digite um numero: ");
    scanf("%d",&num1);
    printf("Digite outro numero: ");
    scanf("%d",&num2);

    //chama a função soma
    printf("Soma de dois numeros %d", soma(num1,num2));
}

//Função soma
int soma(int x, int y) {
    return (x+y);
}
```

Figura 60 – Código Utilizando Função.
Fonte: Autor.

REFERÊNCIAS

Forbellone, A. L. V.; Eberspacher, H. F. **Lógica de Programação**. São Paulo: Makron Books, 2005. Ref. Bib: N.C.

Manzano, J.A.; Oliveira, J.F.; **Algoritmos-Lógica para Desenvolvimento de Programação**; Editora Erica, 1996.

Norton, P. **Introdução à Informática**. Makron Books. 1997.

Schildt, H.; **C Completo e Total**. Makron Books, 1997.

Tenenbaum, A.M. et al; **Estruturas de Dados usando C**; Makron Books, 1995.

Veloso, F. C. **Informática: conceitos básicos**. 7 ed Rio de Janeiro: Campus, 2004.

APÊNDICE A

Atividade Prática I: faça um programa para calcular a autonomia total de um carro e a autonomia para este entrar na reserva.

Dados de entrada:

- taxa (km/l) de consumo do carro;
- quantidade de litros do seu tanque;
- tamanho da reserva em litros.

Saída do Programa:

- autonomia do carro em km;
- autonomia, em km, até chegar a reserva.

Solução da Atividade Prática I (Carro):

```
#include “stdio.h”
#include “stdlib.h”

int main()
{
// Declarando variáveis
float consumo, qtdLitros, reserva;
float autonomia, autonomiaParaReserva;

// Entrada de dados
printf(“Entre com o consumo, Quantidade de Litros e Reserva “);
scanf(“ %f %f %f”, &consumo, &qtdLitros, &reserva);
```

```
// Calculando autonomias
autonomia = consumo * qtdLitros;
autonomiaParaReserva = ((qtdLitros – reserva) * consumo);

// Mostrando resultados
printf("Autonomia %.2f", autonomia);
printf("\n Autonomia para Reserva %.2f", autonomiaParaReserva);
system("PAUSE");
}
```

Atividade Prática II: o objetivo da atividade é a simulação de um semáforo. O programa deve obter uma cor de entrada variando entre verde, amarelo e vermelho e indicar qual seria a próxima cor a aparecer em semáforo.

Entrada:

- solicite do usuário a cor atual.

Saída:

- informe a próxima cor do semáforo.

Obs: As cores devem ser tratadas como constantes const int VERDE=1; const int AMARELO=2; const int VERMELHO=3;

Para praticar bastante faça duas soluções, uma usando o comando if e outra usando o comando switch.

Solução da Atividade Prática II (Semáforo usando if):

```
#include "stdio.h"
#include "stdlib.h"
```

```
int main() {
  // Declarando variáveis
int cor;
const int VERDE=1;  const int AMARELO=2;  const  int
VERMELHO = 3;

// Entrada de dados
printf (“Entre com a cor (1- verde,2-amarelo,3-verme-
lho:”);
scanf(“%d”,&cor);

// Lógica condicional para determinar o resultado
if (cor == VERDE) {
printf(“AMARELO”);
} else if (cor == AMARELO) {
printf(“VERMELHO”);
} else if (cor == VERMELHO) {
printf(“VERDE”);
}
system(“PAUSE”);
}
```

Solução da Atividade Prática II (Semáforo usando switch):

```
#include “stdio.h”
#include “stdlib.h”

int main() {
// Declarando variáveis
int cor;
```

```
const int VERDE=1; const  int AMARELO=2; const int VERMELHO=3;

// Entrada de dados
printf (“Entre com a cor (1- verde,2-amarelo,3-vermelho:”);
scanf(“%d”,&cor);

// Lógica do switch case para determinar o resultado
switch(cor) {
case 1:
printf(“AMARELO”);
break;
case 2:
printf(“VERMELHO”);
break;
case 3:
printf(“VERDE”);
break;
default:
printf(“Cor incorreta!”);
}
system(“PAUSE”);
}
```

Atividade Prática III: faça um programa para verificar se um valor de idade fornecido é maior igual ou menor que 50. Para resolução utilize o operador ternário.

Solução da Atividade Prática III (Operador Ternário):

```
#include “stdio.h”
#include “stdlib.h”
int main()
{
int idade;

printf(“Entre com a idade:”);
scanf(“%d”,&idade);

// IF que verifica se a idade é maior ou igual a 50 anos
idade >= 50 ? printf(“Idade Maior ou igual que 50”) :
printf(“Idade Menor que 50”);

system(“PAUSE”);
}
```

Atividade Prática IV: faça um programa para imprimir os valores de 0 até 100 com incremento de 10, por exemplo (0, 10, 20, 30, ..., 100). Utilize o comando while e do-while em 10. Ver solução no Apêndice.

Solução da Atividade Prática IV (Operador while e do-while):

```
#include “stdio.h”
#include “stdlib.h”

int main()
{
int x = 0;
```

```
// Com ***while*** o programa sempre vai verificar se X é menor ou igual a 100
while (x <= 100) {
printf(" %d ", x);
x = x + 10;
}

// Com o ***do while*** o programa só verifcará após a 2ª iteração, a 1ª sempre será executada
do {
printf(" %d ", x);
x = x + 10;
} while(x <= 100);

system("PAUSE");
}
```

Atividade Prática V: faça um programa que através das entradas de valor, taxa de juros e quantidade de meses, calcule o valor após a aplicação de juros compostos. Use o comando *while* e operadores aritméticos básicos.

- Formula = valor * (1 + taxa / 100)$^{\text{Qtd Meses}}$

Exemplo: valor = 100, taxa = 1 quantidade de meses = 10, o resultado seria = 110.46. Ver solução no Apêndice.

Solução da Atividade Prática V (Juros):

```
#include "stdio.h"
#include "stdlib.h"
int main()
```

```
{
printf("Entre com o valor, juros e quantidade de meses");
scanf(" %f %f %d",&valor,&juros,&meses);

// While para repetir enquanto os meses forem maior que zero
while(meses > 0) {
valor = valor * (1 + (juros / 100));
meses = meses – 1;
}

// Mostrando o resultado para uma variável float
printf("%.2f", valor);

system("PAUSE");
}
```

Atividade Prática VI: similar ao que fizemos com ***while*** e ***do-while***, faça um programa para imprimir os valores de 0 até 100 com incremento de 10, por exemplo (0, 10, 20, 30, ..., 100). Utilize o comando ***for***.

Solução da Atividade Prática VI (Comando for):

```
#include "stdio.h"
#include "stdlib.h"

int main()
{
// for para mostrar na tela a contagem de 0 a 100, só que de 10 em 10
for(int x=0; x <= 100; x = x + 10) {
```

```
// Código para mostrar uma variável inteira (integer) na tela
printf(" %d ", x);
}

system("PAUSE");
}
```

Atividade Prática VII: faça um programa que solicite o número de alunos, objetivando o calculo da média de todo as notas. A cada interação do laço solicite a nota do aluno até finalizar a quantidade de alunos. Por fim, imprima a média. A seguir teremos um exemplo de interação:

– Entre com a quantidade de alunos: 3

– Entre com a nota do Aluno 1: 10.0

– Entre com a nota do Aluno 2: 5.0

– Entre com a nota do Aluno 3: 8.0

– Media da Turma = 7.67

Solução da Atividade Prática VII (Média dos Alunos):

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

int main()
{
int qtdAlunos = 0;

// Entrada de dados
printf("Qual a quantidade de Alunos? ");
scanf("%d", &qtdAlunos);
```

```
float alunos[qtdAlunos];
int i;

// Entrando com as notas dos alunos
for (i = 0; i < qtdAlunos; i++) {
printf("\nQual a nota do Aluno %d? ", (i+1));
scanf("%f", &alunos[i]);
}

// Somando as notas dos alunos
float media = 0;
for (i = 0; i < qtdAlunos; i++) {
media = media + alunos[i];
}

// Calculando a média dos alunos
media = media / qtdAlunos;
printf("\nA média da turma é: %.2f \n", media);

system("PAUSE");
}
```

Atividade Prática VIII (Matrizes Unidimensionais): faça um programa para carregar uma matriz com 100 elementos com o seguinte domínio: (-1,-2,...,-100). Ou seja: x[0]=-1, x[1]=-2, ..., x[99]= -100.

Solução Atividade Prática VIII (Matrizes Unidimensionais):

```
#include "stdio.h"
#include "stdlib.h"
```

```
int main()
{
// Declarando variáveis
int x[100];
int z=0;
int y;

// Acumulando valores de -1 a -100
for(y = -1; y >= -100; y--) {
x[z]=y;
z++;
}

printf(" Primeiro Termo=%i Termo do Meio=%i e Ultimo
Temo=%i",x[0],x[49],x[99]);

system("PAUSE");
}
```

Atividade Prática IX (Maior Valor): faça um programa para carregar uma matriz com uma lista de preços {200, 1000, 3000, 400 e 600}.

Por fim, dinamicamente ache e imprima o elemento que possui o maior preço.

Solução da Atividade Prática IX (Maior Valor):

```
#include "stdio.h"
#include "stdlib.h"
```

```
int main()
{

// Declarando variáveis
float preco[5];
float maiorValor;
preco[0]=200;
preco[1]=1000;
preco[2]=3000;
preco[3]=400;
preco[4]=600;
int i;
maiorValor = preco[0];

// Verificando o maior valor
for(i=1; i<5; i++) {
if (maiorValor < preco[i]) {
maiorValor = preco[i];
}
}
printf("Maior Preco %f",maiorValor);
system("PAUSE");
}
```

Atividade Prática X (Funções): faça um programa que calcule a autonomia total e autonomia até chegar a reserva de um carro. A entrada será o consumo e a quantidade de litros. Crie uma função para cada tipo de autonomia calculada, exemplo:

```
float kmTotal(float consumo , float qtdLitros) {
```

```
// código
}
float kmParaReserva(float consumo , float qtdLitros) {
// código
}
```

Solução da Atividade Prática X (Funções):

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

// Declarando variáveis globais
float km(float consumo , float qtdLitros);
float kmParaReserva(float consumo , float qtdLitros);

int main() {
float consumo,qtdLitros;
printf("Entre com o consumo e a quantidade de litros do
carro: ");
scanf("%f %f", &consumo, &qtdLitros);

printf("Autonomia total de %.2fkm e ate a reserva de %.2f
", km(consumo,qtdLitros), kmParaReserva(consumo,qtd
Litros));

system("PAUSE");
}

// Método para calcular a Quilometragem
float km(float consumo , float qtdLitros) {
```

```
return consumo * qtdLitros;
}

// Método para calcular a Quilometragem para a reserva
float kmParaReserva(float consumo , float qtdLitros) {
return km(consumo, qtdLitros) – 8*consumo;
}
```

Impressão e acabamento

egba
EMPRESA GRÁFICA DA BAHIA

Rua Mello Moraes Filho, nº 189, Fazenda Grande do Retiro
CEP: 40.352-000 – Tels.: (71) 3116-2837/2838/2820
Fax: (71) 3116-2902
Salvador-Bahia
E-mail: encomendas@egba.ba.gov.br