

Algoritmos Computacionais com aplicações em C

Bacharelado em Sistemas de Informação

Engenharia Mecatrônica

IF Sudeste MG

Filippe jabour

26 de março de 2012

<http://www.jabour.com.br>

Este material pode ser usado livremente, copiado ou distribuído, desde que citada a autoria.

Feito no L^AT_EX em ambiente Linux.

Sumário

Lista de Algoritmos/Programas	vii
Lista de figuras	x
Lista de tabelas	xi
1 Conceitos Iniciais	1
1.1 Referências e bibliografia	1
1.2 Algoritmos e a linguagem C	1
1.3 Variável	2
1.4 Tipos de variáveis	3
1.5 Conceitos iniciais: aplicações em C	3
1.5.1 O conceito de função	3
1.5.2 A função main	4
1.5.3 Primeiros exemplos de programas em C	4
1.6 Exercícios Propostos	6
1.6.1 Média aritmética	7
1.6.2 Média ponderada	7
1.6.3 Aumento de salário	7
1.6.4 Salário base com gratificação e impostos	7
1.6.5 Operações com um número	7

1.6.6	Preço do carro, impostos e lucro	7
1.6.7	Cálculo de salário	8
1.6.8	Saldo da conta após emissão de cheques	8
1.6.9	Consumo de ração de gatos	8
1.6.10	Medida da escada enconstada	8
2	O comando if-else (se-senão)	9
2.1	Descrição e exemplos	9
2.2	if-else : aplicações em C	12
2.2.1	A função scanf	12
2.2.2	Uso do se-senão (if-else) em C	13
2.3	Exercícios Propostos	15
2.3.1	Adivinhar um número	15
2.3.2	Verificar se um número é par ou ímpar	15
2.3.3	Achar o maior de 3 números	16
2.3.4	Verificar se um número é positivo, negativo ou nulo	16
2.3.5	Equação do segundo grau	16
2.3.6	Soma números e escolhe uma operação com base no resultado da soma	16
2.3.7	Verifica se empréstimo pode ser concedido	16
3	Switch-case (escolha-caso)	17
3.1	Descrição e exemplos	17
3.2	switch-case : aplicações em C	20
3.3	Exercícios Propostos	21
3.3.1	Peso nos planetas	21
3.3.2	Questão de múltipla escolha	21
3.3.3	Calculadora simples	22

4 For (para)	23
4.1 Descrição e exemplos	23
4.2 for : aplicações em C	26
4.3 Exercícios Propostos	28
4.3.1 Adivinhar um número com n tentativas	28
4.3.2 Menu de opções e um for	28
4.3.3 Progressão aritmética	28
4.3.4 Sequência de Fibonacci	29
4.3.5 Par de números com soma definida	29
4.3.6 Soma de alguns números digitados	29
4.3.7 Achar os divisores de um número	29
4.3.8 Imprimir somente acima da diagonal principal	29
5 while e do-while (enquanto e faça-enquanto)	31
5.1 Descrição e exemplos	31
5.2 while e do-while : aplicações em C	33
5.3 Exercícios Propostos	35
5.3.1 Adivinhar n° até o usuário acertar ou desistir	35
5.3.2 Preço de passagem aérea	35
5.3.3 Lê n°s e exibe o maior, o menor e a quantidade	36
5.3.4 Menu para somar ou multiplicar n°s	36
5.3.5 Achar o número de múltiplos em um intervalo	36
6 String, vetor e matriz	37
6.1 Vetor	37
6.2 String	39
6.2.1 Funções de manipulação de strings	40

strlen()	40
strcpy()	41
strcat()	42
strcmp()	42
6.3 Matriz	43
6.3.1 Matrizes de strings	45
6.4 Exercícios Propostos	46
6.4.1 Calcular a média dos elementos de um vetor	46
6.4.2 Gerar vetor soma e produto a partir de 2 vetores originais	46
6.4.3 Imprimir vetor de trás pra frente	46
6.4.4 Concatenar 2 vetores	47
6.4.5 Achar maior e menor elementos do vetor e somar seus elementos	47
6.4.6 Confere senha digitada	47
6.4.7 Imprimir uma string de trás para frente	47
6.4.8 Lê e imprime strings e números	47
6.4.9 Soma de matrizes	47
6.4.10 Soma dos elementos da diagonal principal	48
6.4.11 Menu e matriz	48
7 Funções	49
7.1 Funções em C	49
7.2 Regras de escopo	52
7.3 Exercícios Propostos	53
7.3.1 Calculadora usando funções	53
7.3.2 Equação do segundo grau usando funções	53
7.3.3 Criação de um vetor aleatório usando função	53

7.3.4	Concatenação de 2 vetores ordenados em um 3º também ordenado	53
7.3.5	Achar o maior, menor, média, soma e produto dos elementos do vetor, usando funções . . .	54
7.3.6	Produto de matrizes	54
7.3.7	Ordenação de vetor	54
8	struct (Estrutura)	55
8.1	struct : conceitos iniciais	55
8.2	Acessando os elementos (campos) da estrutura	57
8.3	Vetores de estruturas	57
8.4	Exercícios Propostos	57
8.4.1	Cadastro de produtos	57
9	Solução dos Exercícios	58
9.1	Respostas dos exercícios do Capítulo 1	58
9.2	Respostas dos exercícios do Capítulo 2	58
9.3	Respostas dos exercícios do Capítulo 3	63
9.4	Respostas dos exercícios do Capítulo 4	66
9.5	Respostas dos exercícios do Capítulo 5	72
9.6	Respostas dos exercícios do Capítulo 6	73
9.7	Respostas dos exercícios do Capítulo 7	73
	Referências Bibliográficas	79

Lista de Algoritmos/Programas

1.1	Algoritmo para somar 2 números	1
1.2	Algoritmo c/ declaração de tipos	3
1.3	Programa vazio com include e main	4
1.4	Programa que cria e imprime uma variável inteira	4
1.5	Programa que soma dois números	5
1.6	Programa que multiplica dois números	6
1.7	Cálculo da área do triângulo	6
2.1	O comando se	9
2.2	Uso do comando se	10
2.3	O comando se-senão	10
2.4	Uso do comando se-senão	11
2.5	Uso do comando se-senão	11
2.6	se-senão aninhados	12
2.7	Um exemplo simples com scanf	13
2.8	Um if simples	13
2.9	if-else simples	13
2.10	if-else com aninhamento	14
2.11	if-else com aninhamento	14
3.1	O comando escolha-caso	17

3.2	Exemplo simples com escolha-caso	18
3.3	Cálculo do peso em outros planetas	19
3.4	O comando switch-case	20
4.1	Um exemplo simples com o laço para	23
4.2	Imprimir os números pares de 0 a 1000	24
4.3	para com decremento da variável	24
4.4	Algoritmo 4.3 implementado em C	25
4.5	Imprimir os ímpares de 30 a 100	26
4.6	3 variáveis mudando a cada iteração	26
4.7	Um for dentro do outro	27
5.1	Um exemplo simples com o laço enquanto	31
5.2	Um exemplo simples com o laço faça-enquanto	32
5.3	Uso do while combinado com um menu de opções	33
6.1	O primeiro exemplo com vetores	38
6.2	Variável como índice do vetor	39
6.3	Um exemplo simples de string e gets()	40
6.4	A função strlen()	41
6.5	A função strcpy()	41
6.6	A função strcat()	42
6.7	A função strcmp()	43
6.8	Um exemplo simples com matriz	44
6.9	Lê e exibe um texto com várias linhas	45
7.1	Programa com a função calculaMedia	49
7.2	Passando uma matriz como argumento da função	51
9.1	Resposta do exercício 1.6.10	58
9.2	Resposta do exercício 2.3.1	58

9.3	Resposta do exercício 2.3.2	59
9.4	Resposta do exercício 2.3.3	59
9.5	Resposta do exercício 2.3.4	60
9.6	Resposta do exercício 2.3.5	61
9.7	Resposta do exercício 2.3.6	62
9.8	Resposta do exercício 2.3.7	62
9.9	Resposta do exercício 3.3.1	63
9.10	Resposta do exercício 3.3.2	64
9.11	Resposta do exercício 3.3.3	65
9.12	Resposta do exercício 4.3.1	66
9.13	Resposta do exercício 4.3.2	67
9.14	Resposta do exercício 4.3.3	68
9.15	Resposta do exercício 4.3.4	69
9.16	Resposta do exercício 4.3.5	69
9.17	Resposta do exercício 4.3.6	70
9.18	Resposta do exercício 4.3.7	70
9.19	Resposta do exercício 4.3.8	71
9.20	Resposta do exercício 5.3.1	72
9.21	Resposta do exercício 6.4.1	73
9.22	Resposta do exercício 7.3.4	73
9.23	Resposta do exercício 7.3.6	75
9.24	Resposta do exercício 7.3.7	76

Lista de Figuras

1.1	O ciclo de desenvolvimento, compilação e execução de um programa em C	2
-----	---	---

Lista de Tabelas

2.1	Operadores relacionais	10
2.2	Operadores lógicos	10
3.1	Gravidades relativas em outros planetas	19
4.1	Saída do programa 4.4	25
5.1	Preço de passagens aéreas por região	35

Capítulo 1

Conceitos Iniciais

1.1 Referências e bibliografia

Esta apostila usou como referência e bibliografia as seguintes obras, sítios e materiais: C - Completo e Total [3], Introdução à programação - 500 algoritmos resolvidos [1], Turbo C: guia do usuário [2].

1.2 Algoritmos e a linguagem C

Algoritmo é uma sequência finita de passos com o objetivo de solucionar um problema.

Dado um problema ou tarefa qualquer, você pode propor uma lista de ações sequencias que, se executadas, resolverão o problema ou executarão a tarefa.

Se estivermos tratando de computação, estes passos deverão ser apropriados para a execução em um computador.

Para ser executado em um computador, seu algoritmo precisa ser traduzido (ou reescrito) em uma linguagem de programação. Nesta apostila utilizaremos a linguagem C.

Uma vez escrito o programa em C, o mesmo precisa se traduzido para a linguagem do computador para ser executado. Esta tradução se chama compilação (Figura 1.1).

A cada modificação do programa inicial, chamado de programa fonte ou código fonte, a compilação deve ser refeita para gerar um novo programa executável que reflita as modificações feitas.

Vejam os exemplos no algoritmo 1.1

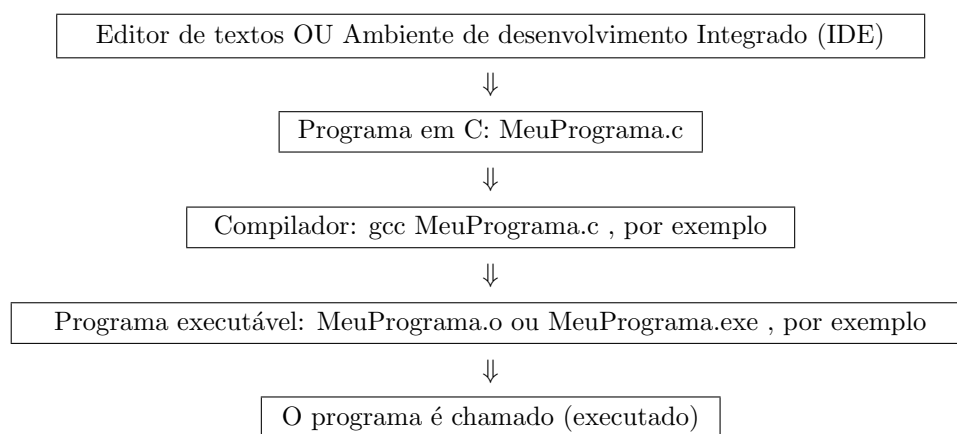


Figura 1.1: O ciclo de desenvolvimento, compilação e execução de um programa em C

```

1 início
2     A = 10;
3     B = 23;
4     C = A + B;
5     Mostrar o valor de C;
6 fim
  
```

A e B são chamadas variáveis, como na matemática, e recebem valores numéricos (10 e 23, respectivamente). A e B são somadas e o resultado é gravado (armazenado) em outra variável chamada C . Por fim, o resultado da soma é mostrado ou exibido para uma pessoa (ou usuário) qualquer.

Em geral, um programa de computador tem uma entrada, ou dados de entrada, executa uma computação (“faz alguma coisa com estes dados”) e exibe uma saída.

No exemplo do algoritmo 1.1, podemos adaptar os comandos $A = 10$; e $B = 23$; como a entrada de dados. $C = A + B$; é a computação em si (executar uma soma, no caso). Por fim, o enunciado **Mostrar o valor de C** ; é a saída do programa (mostrar ao usuário o valor da soma, no caso, 33).

1.3 Variável

Computadores executam operações matemáticas e comparações de valores no processador (CPU). Os valores usados nas operações matemáticas e nas comparações ficam armazenados (gravados) na memória principal. Para saber o que está aonde, a memória é dividida em células e estas células possuem endereços (como o número da sua casa).

Uma variável é o nome dado a um endereço de memória ou, em outras palavras, uma variável é um endereço de memória.

Como vimos no algoritmo 1.1, variáveis podem receber valores ($A = 10$, $B = 23$). Este comando é chamado

atribuição de valores.

1.4 Tipos de variáveis

As variáveis podem ser de diversos **tipos**. Um deles é o tipo numérico. Variáveis numéricas podem ser, por exemplo, do tipo **inteiro** (**int**) (-3 , 0 , 4 , 8) ou do tipo **real** (**float**) (3.14 , -5.67 , 9 , -4 , 0).

Vejam o algoritmo 1.1 com a declaração explícita do tipo das variáveis (algoritmo 1.2)

Algoritmo/Programa 1.2: Algoritmo c/ declaração de tipos

```
1 início
2     int x;
3     int y;
4     int soma;
5     x = 8;
6     y = 25;
7     soma = x + y;
8     imprimir(soma);
9 fim
```

1.5 Conceitos iniciais: aplicações em C

Uma vez feito o algoritmo, você pode optar por testá-lo em um computador. Para isto, ele deve ser reescrito em uma linguagem de programação. Dizemos que o algoritmo será implementado nesta linguagem. Como dito na Seção 1.2, usaremos a linguagem C.

1.5.1 O conceito de função

Uma **função** é um trecho de código (um programa) já pronto que pode ser utilizado dentro do seu programa. Para usar uma função pronta, basta escrever o nome dela no seu código. Dizemos que estamos **chamando a função**.

Para usar estas funções já prontas, precisamos “avisar ao C” (mais especificamente avisar ao compilador C) que esta função será usada. Isto é feito **incluindo bibliotecas** ao programa. Bibliotecas são conjuntos de funções prontas.

O comando em C para fazer isto é:

```
#include <nome_da_biblioteca>
```

Nos nossos primeiros programas, usaremos apenas a biblioteca básica de entrada e saída chamada **stdio.h**.

Deste modo, todos os programas começarão com a inclusão desta biblioteca:

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

1.5.2 A função main

Todo programa precisa ter um ponto de partida. Um ponto a partir do qual ele será iniciado. Em outras palavras, a primeira instrução a ser executada. No caso do C, o programa começa pela **função principal**, chamada **função main**.

A forma de chamá-la é a seguinte:

```
int main()
```

Em seguida, abre-se uma chave ({) para indicar o início do **bloco** e no fim da função **main** esta chave é fechada (}) para indicar o fim do bloco (ou o fim da função).

Algoritmo/Programa 1.3: Programa vazio com include e main

```
1 #include <stdio.h>
2
3 int main(){
4     . . .
5     . . .
6 }
```

1.5.3 Primeiros exemplos de programas em C

Vamos então ao primeiro programa em C, o programa 1.4:

Algoritmo/Programa 1.4: Programa que cria e imprime uma variável inteira

```
1 #include <stdio.h>
2
3 int main(){
4     int a;
5     a = 10;
6     printf("O valor de a é: %d ",a);
7     system("pause");
8 }
```

O Programa 1.4 começa com a declaração (linha 4) e inicialização (atribuição de um valor) à variável **a** (linha 5). Em seguida, é chamada a função **printf** que corresponde ao **imprimir** do algoritmo.

A função **printf** imprime um texto fixo (entre aspas duplas (“ ”)). No caso, o texto é **O valor de a é: O**. O **printf** pode imprimir ainda valores de variáveis (depois da vírgula), no caso o valor de **a**. O **%d** informa que

a é uma variável do tipo inteiro. Além disso, o valor da variável **a** será colocado no mesmo local da frase onde se encontra o **%d**.

Por fim, a função **system** chamada com o argumento “**pause**” (linha 7), usada no Windows, faz com que o programa pare a execução e peça que qualquer tecla seja digitada. Serve para que a janela de execução do programa não se feche e o usuário possa ver o resultado final (ou a saída) do programa. Este comando não será mais incluído nos exemplos e respostas dos exercícios. Fica a critério de cada um o seu uso, quando necessário.

É possível inserir **comentários** ao longo de um programa. São textos que você escreve para explicar o funcionamento e documentar o desenvolvimento do programa. Eles **não tem nenhuma influência** sobre a execução do programa. Para um comentário de uma linha, basta iniciar com **//**. Pode-se usar **//** no fim da linha para que, a partir daí, exista um comentário. Comentários maiores, começam com **/*** e terminam com ***/**, podendo ter várias linhas.

```
// Comenta linha
```

```
/* Comenta
```

```
todo
```

```
um
```

```
trecho */
```

É recomendável que nomes de variáveis representem o significado das mesmas. Use **raizUm** em lugar de **r**; **notaDoAluno** em lugar de **n**; etc. Insira tantos comentários quantos julgar necessários.

O Programa 1.5 traz algumas novidades com relação ao 1.4. As 3 variáveis são declaradas todas na mesma linha (9), é feita uma soma e 3 valores são impressos pela função **printf** (13).

Algoritmo/Programa 1.5: Programa que soma dois números

```
1  /* Programa desenvolvido pelo Prof. Jabour
2  em março de 2011.
3  O programa soma dois números, exibe a soma e
4  o resultado */
5
6  #include <stdio.h>
7
8  int main(){
9      int a, b, c; //declaração de variáveis
10     a = 10;
11     b = 32;
12     c = a + b; //execução do cálculo
13     printf("\n%d + %d = %d\n",a,b,c); //impressão do resultado
14 }
```

Os operadores aritméticos são relacionados a seguir:

+ ⇒ Adição

- ⇒ Subtração

* ⇒ Multiplicação

/ ⇒ Divisão

O Programa 1.6 declara 3 números reais (**float**). Dois deles recebem valores (são inicializados) e o terceiro recebe o produto dos 2 primeiros (*). A **máscara %f** informa que o número a ser impresso é real e o **.1** faz com que o número seja impresso com uma casa decimal.

O comando `\n` faz pular uma linha. É como um **enter**. Chamamos de **quebra de linha**.

Algoritmo/Programa 1.6: Programa que multiplica dois números

```
1 #include <stdio.h>
2
3 int main( ){
4     float a, b, c;
5     a = 8.4;
6     b = 4.5;
7     c = a * b;
8     printf("\n%.1f x %.1f = %.1f\n",a,b,c);
9 }
```

Mais um exemplo (Programa 1.7). Fazer um programa que calcule a área do triângulo (dadas a base e a altura).

Algoritmo/Programa 1.7: Cálculo da área do triângulo

```
1 #include <stdio.h>
2
3 int main(){
4     float b, h, area;
5     b = 10;
6     h = 3.5;
7     area = b * h / 2;
8     printf("\nbase = %f; altura = %f; area = %.3f\n",b,h,area);
9 }
```

1.6 Exercícios Propostos

Como ainda não estudamos como inserir dados via teclado (digitar dados de entrada), considere nestes exercícios que a entrada de dados será feita declarando variáveis e atribuindo valores no próprio código fonte.

Deste modo, quando os enunciados falarem em “receber valores, notas, salário, números, etc”, crie as variáveis e atribua os valores a elas.

1.6.1 Média aritmética

Faça um programa que receba três notas, calcule e mostre a média aritmética entre elas.

1.6.2 Média ponderada

Faça um programa que receba três notas e seus respectivos pesos, calcule e mostre a média ponderada dessas notas.

1.6.3 Aumento de salário

Faça um programa que receba o salário de um funcionário e o percentual de aumento, calcule e mostre o valor do aumento e o novo salário.

1.6.4 Salário base com gratificação e impostos

Faça um programa que receba o salário-base de um funcionário, calcule e mostre o salário a receber, sabendo-se que esse funcionário tem gratificação de 5% sobre o salário-base e paga imposto de 7% sobre salário-base.

1.6.5 Operações com um número

Faça um programa que receba um número positivo e maior que zero, calcule e mostre:

1. O número digitado ao quadrado;
2. O número digitado ao cubo;
3. A metade do número digitado;
4. O sucessor do número digitado.

1.6.6 Preço do carro, impostos e lucro

O custo ao consumidor de um carro novo é a soma do preço de fábrica com o percentual de lucro do distribuidor e dos impostos aplicados ao preço de fábrica. Faça um programa que receba o preço de fábrica de um veículo, o percentual de lucro do distribuidor e o percentual de impostos. Calcule e mostre:

- O valor correspondente ao lucro do distribuidor;
- O valor correspondente aos impostos;
- O preço final do veículo.

1.6.7 Cálculo de salário

Faça um programa que receba o número de horas trabalhadas e o valor do salário mínimo. Calcule e mostre o salário a receber seguindo as regras abaixo:

- A hora trabalhada vale a metade do salário mínimo;
- O salário bruto equivale ao número de horas trabalhadas multiplicado pelo valor da hora trabalhada;
- o imposto equivale a 3% do salário bruto;
- O salário a receber equivale ao salário bruto menos o imposto.

1.6.8 Saldo da conta após emissão de cheques

Um trabalhador recebeu seu salário e o depositou em sua conta corrente bancária. Esse trabalhador emitiu dois cheques e agora deseja saber seu saldo atual. Sabe-se que cada operação bancária de retirada paga CPMF de 0,38% e o saldo inicial da conta está zerado. Faça um programa que receba o valor do salário e dos dois cheques emitidos e calcule o saldo atual.

1.6.9 Consumo de ração de gatos

Pedro comprou um saco de ração com peso em quilos. Pedro possui dois gatos para os quais fornece a quantidade de ração em gramas. Faça um programa que receba o peso do saco de ração e a quantidade de ração fornecida para cada gato. Calcule e mostre quanto restará de ração no saco após cinco dias.

1.6.10 Medida da escada enconstada

Faça um programa que receba a medida do ângulo formado por uma escada apoiada no chão e encostada na parede (ângulo entre a escada e o chão) e a altura da parede onde está a ponta da escada. Calcule e mostre a medida desta escada.

Resposta: programa 9.1.

Capítulo 2

O comando if-else (se-senão)

2.1 Descrição e exemplos

É comum termos que tomar decisões ao longo do algoritmo. Imagine que, se um valor for positivo, uma ação será tomada. Se ele for zero ou negativo, outra ação é necessária. Este tipo de decisão é representado pelo comando **se**.

Vamos ver como é a sintaxe do **se** (Algoritmo 2.1):

Algoritmo/Programa 2.1: O comando **se**

```
1 se (condição) {  
2     comando 1;  
3     comando 2;  
4     . . .  
5     comando n;  
6 }
```

A **condição** entre parênteses (depois do **se**) é avaliada. Se o resultado for **verdadeiro**, o programa executa o bloco de instruções entre chaves, que vem em seguida (comando 1, comando 2, etc., comando n). Se for falso, o programa simplesmente pula este bloco e continua a execução após o fechamento da chave correspondente.

Assim, se a condição for ($a < 30$), por exemplo, caso o valor de **a** seja 23, o resultado será **verdadeiro** e os comandos dentro do **se** serão executados. Dizemos que o programa “entrou no **se**”. Se **a** valer 234, por exemplo, a condição será **falsa** e o programa pulará o bloco entre chaves depois do **se**. Dizemos que o programa “não entrou no **se**”.

Resumindo, condição assume um **valor lógico (verdadeiro ou falso)**. **Se** for verdadeiro, executa-se o interior das chaves do **se** for falso, não.

A Tabela 2.1 mostra os operadores relacionais e a Tabela 2.2 os operadores lógicos.

Operador	Significado
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual
==	igual
!=	diferente (“não igual”)

Tabela 2.1: Operadores relacionais

Operador	Significado
&&	“e”
	“ou”
!	negação (não)

Tabela 2.2: Operadores lógicos

Exemplo: Fazer um algoritmo que lê um valor inteiro e imprime o valor caso ele seja maior que 10.

Solução (Algoritmo 2.2)

Algoritmo/Programa 2.2: Uso do comando **se**

```

1 início
2     int valor;
3     ler(valor);
4     se( valor > 10 ) {
5         imprima(valor)
6     }
7 fim
```

A forma geral do comando **se**, incorpora também um **senão**. Deste modo, a **condição** é avaliada. Se for **verdadeira**, procede-se como descrito acima. Se for **falsa**, o bloco após o **senão** é executado. Veja a estrutura no Algoritmo 2.3:

Algoritmo/Programa 2.3: O comando **se-senão**

```

1 se (condição) {
2     comando 1;
3     comando 2;
4     . . .
5     comando n;
6 }
7 senão {
```

```

8      comando n+1;
9      comando n+2;
10     . . .
11     comando n+m;
12 }

```

Exemplo (Algoritmo 2.4): Fazer um algoritmo que receba a idade de uma pessoa e informe se ela é maior ou menor de idade.

Algoritmo/Programa 2.4: Uso do comando **se-senão**

```

1  início
2      int idade;
3      ler(idade);
4      se ( idade >= 18) {
5          imprima("É maior de idade");
6      }
7      senão {
8          imprima("É menor de idade");
9      }
10 fim

```

Outro exemplo (Algoritmo 2.5): Faça um algoritmo que leia o salário base do empregado e o seu total de vendas. Em seguida é calculado o salário final do empregado. O empregado que vendeu mais de R\$ 1000 recebe um bônus de 20% em seu salário final, calculado sobre o salário base. O total de vendas não se incorpora ao salário final, sendo usado apenas como critério de distribuição do bônus.

Algoritmo/Programa 2.5: Uso do comando **se-senão**

```

1  início
2      float salarioBase , salarioFinal , vendas;
3      ler(salarioBase);
4      ler(vendas);
5      se (vendas > 1000) {
6          salarioFinal = salarioBase * 1.2;
7      }
8      senão {
9          salarioFinal = salarioBase;
10     }
11     imprima(salarioFinal);
12 fim

```

É possível que, dentro de um bloco do **se** e/ou do bloco **senão**, tenhamos outro ou outros **se-senão**. É o que chamamos de aninhamento de **se**. Vejamos através de um exemplo, no Algoritmo 2.6.

Considere o critério de aprovação no IF Sudeste MG. Quando o aluno possui menos de 75% de frequência, ele está reprovado, independente da nota. Possuindo frequência suficiente, são 3 as possibilidades. Nota igual

ou superior a 60 aprova o aluno. Nota entre 40 (inclusive) e 59 o coloca em prova final e nota menor que 40 o reprova. A “dupla reprovação” (frequência menor que 75% e nota menor que 40) é registrada como “reprovação por infrequência”). Faça o algoritmo que leia nota e frequência e informe o resultado final do aluno.

Algoritmo/Programa 2.6: **se-senão** aninhados

```
1 //alg008.txt
2
3 início
4     float nota, frequencia;
5     ler(nota);
6     ler(frequencia);
7     se( frequencia < 75 ) {
8         imprima("reprovado por infrequência");
9     }
10    senão {
11        se ( nota >= 60 ) {
12            imprima("aprovado");
13        }
14        senão {
15            se ( nota >= 40 ) {
16                imprima("prova final");
17            }
18            senão {
19                imprima("reprovado por nota");
20            }
21        }
22    }
23 fim
```

Material desenvolvido por Filippé Jabour - <http://www.jabour.com.br>.

2.2 if-else: aplicações em C

2.2.1 A função scanf

Para ler caracteres a partir do teclado usamos a função **scanf**. Ela necessita da máscara aos mesmos moldes da **printf**. Outro argumento passado à função é o endereço da variável que receberá o que foi digitado.

Exemplos:

scanf("%d",&var) → recebe uma variável inteira (%d) digitada via teclado e grava o valor digitado no endereço (&) da variável **var**. Assim, se você digitar 34 e teclar **enter**, a variável **var** passará a ter o valor 34.

2.2.2 Uso do se-senão (if-else) em C

Primeiro vamos dar um exemplo muito simples, apenas com o uso do **scanf** (Programa 2.7).

Algoritmo/Programa 2.7: Um exemplo simples com scanf

```
1 #include <stdio.h>
2
3 int main(){
4     int a;
5     printf("\nDigite um valor para a: ");
6     scanf("%d",&a); //nesta linha, o valor digitado é gravado no endereço de a
7     printf("\nVocê digitou: %d\n",a);
8 }
```

Agora, no Programa 2.8, o **scanf** com um **if** simples.

Algoritmo/Programa 2.8: Um if simples

```
1 #include <stdio.h>
2
3 int main(){
4     int a;
5     printf("\nDigite um número: ");
6     scanf("%d",&a);
7     if(a>10) {
8         printf("\n%d é maior que 10.\n",a);
9     }
10 }
```

Programa 2.9, com **if-else**.

Algoritmo/Programa 2.9: if-else simples

```
1 #include <stdio.h>
2
3 int main(){
4     int a;
5     printf("\nDigite um número: ");
6     scanf("%d",&a);
7     if(a>10) {
8         printf("\n%d é maior que 10.\n",a);
9     }
10    else {
11        printf("\n%d é menor ou igual a 10.\n",a);
12    }
13 }
```

Exemplo (Programa 2.10): Leia a idade da pessoa e direcione para uma fila de idoso quando for maior que 60 anos; informe que não pode entrar quando for menor de 18 anos e direcione para a fila comum quando for maior de idade e menor que 60 anos.

Algoritmo/Programa 2.10: **if-else** com aninhamento

```
1 //prog008.c
2
3 #include <stdio.h>
4
5 int main(){
6     int idade;
7     printf("\nDigite a idade: ");
8     scanf("%d",&idade);
9     if( idade >= 60 ) { //observe o uso de maior ou igual
10         printf("\nDirija-se à fila preferencial.\n");
11     }
12     else {
13         if (idade >= 18) {
14             /* Como estamos no else de "idade >= 60",
15             a idade já é "< 60". Entrando neste if, ela
16             também é maior de idade. Logo:
17             */
18             printf("\nDirija-se à fila comum.\n");
19         }
20         else {
21             printf("\nEntrada proibida.\n");
22         }
23     }
24 }
```

Veja no Programa 2.11 a implementação em C do Algoritmo 2.6.

Algoritmo/Programa 2.11: **if-else** com aninhamento

```
1 //prog009.c
2
3 #include <stdio.h>
4
5 int main() {
6     float nota, frequencia;
7     printf("\nNota: ");
8     scanf("%f",&nota);
9     printf("\nFrequência: ");
10    scanf("%f",&frequencia);
11    if ( frequencia < 75 ) {
12        printf("\nReprovado por infrequência.\n");
13    }
14    else {
```

```
15         if ( nota >= 60 ) {
16             printf("\nAprovado\n");
17         }
18         else {
19             if ( nota >= 40 ) {
20                 printf("\nProva final.\n");
21             }
22             else {
23                 printf("\nReprovado por nota.\n");
24             }
25         }
26     }
27 }
```

2.3 Exercícios Propostos

2.3.1 Adivinhar um número

Faça um programa que gere um número aleatório de 0 a 9, receba um palpite via teclado e informe se o palpite é certo ou errado.

Dicas:

- A linha `srand (time(NULL));` faz com que o número gerado varie a cada execução do programa.
- A linha `numeroGerado = rand();` gera um número aleatório (função `rand()`) e atribui este valor à variável `numeroGerado`.
- O operador `%` é chamado **operador módulo** ou **resto da divisão**. Assim, `x = y % 10;` atribui à variável `x` o resto da divisão de `y` por 10. Ou seja, se `y = 23`, `x` receberá o valor 3.
- O número gerado com a função `rand()` é grande. Se aplicarmos o operador **módulo** deste número grande por 10, por exemplo, teremos sempre o resto da divisão por 10, o que será um número de zero a 9.
- Teste de igualdade em C é feito com `==` e não apenas com um `=`.

Resposta: programa 9.2. Não olhe a resposta antes de tentar várias vezes, antes de discutir com os colegas, procurar o professor, o monitor, etc.

2.3.2 Verificar se um número é par ou ímpar

Faça um programa que receba um número via teclado e verifique se este número é par ou ímpar.

Resposta: programa 9.3.

2.3.3 Achar o maior de 3 números

Faça um programa que receba 3 números via teclado e imprima o maior deles.

Resposta: programa 9.4.

2.3.4 Verificar se um número é positivo, negativo ou nulo

Ler um número e informar se ele é positivo, negativo ou nulo.

Resposta: programa 9.5.

2.3.5 Equação do segundo grau

Faça um programa que receba os 3 coeficientes e encontre a(s) raiz(es) reais da equação do segundo grau. Não aceite o coeficiente de x^2 igual a zero.

Resposta: programa 9.6.

2.3.6 Soma números e escolhe uma operação com base no resultado da soma

Faça um programa que recebe dois números e efetua a adição. Caso o valor somado seja maior que 20, este deverá ser apresentado somando-se a ele 8; caso o valor somado seja menor ou igual a 20, este deverá ser apresentado subtraindo-se 5.

Resposta: programa 9.7.

2.3.7 Verifica se empréstimo pode ser concedido

Uma empresa abriu uma linha de crédito para os funcionários. O valor da prestação não pode ultrapassar 30% do salário. Faça um programa que receba o salário, o valor do empréstimo e o número de prestações e informe se o empréstimo pode ser concedido. Nenhum dos valores informados pode ser zero ou negativo.

Resposta: programa 9.8.

Capítulo 3

Switch-case (escolha-caso)

3.1 Descrição e exemplos

O comando **if-else** permite uma decisão que direcione o fluxo de execução do algoritmo em dois caminhos diferentes. O programa segue para o **if** ou para o **else**.

Existe um comando que tem múltiplos caminhos possíveis. Trata-se do comando **escolha-caso**. Existem algumas opções possíveis e o programa executa **uma delas**, de acordo com uma comparação de igualdade feita.

A estrutura do **escolha-caso** é dada no algoritmo 3.1.

Algoritmo/Programa 3.1: O comando **escolha-caso**

```
1 //alg009.txt
2
3 escolha (variável) {
4     caso valor_1:
5         comando_1;
6         comando_2;
7         . . .
8         comando_n;
9         pare;
10    caso valor_2:
11        comando_n+1;
12        comando_n+2;
13        . . .
14        comando_n+k;
15        pare;
16    caso valor_3:
17        comando_n+k+1;
18        comando_n+k+2;
19        . . .
```

```

20         comando_n+k+m;
21         pare;
22     . . .
23     . . .
24     default:
25         comando_n+k+m+y+1;
26         comando_n+k+m+y+2;
27         . . .
28         comando_n+k+m+y+x;
29 }

```

O valor da **variável** entre parênteses, à frente da palavra **escolha**, é comparado a cada **valor_i** à frente de cada **caso**. Ao ser encontrado um valor **igual** (**apenas a igualdade é testada**), a sequência de comandos a seguir é executada até o comando **pare**.

Vamos ao primeiro exemplo (algoritmo 3.2):

Algoritmo/Programa 3.2: Exemplo simples com **escolha-caso**

```

1  início
2      int x;
3
4      escreva("Digite um número");
5      ler(x);
6
7      escolha (x) {
8          caso 1:
9              imprima("Você digitou 1");
10             pare;
11         caso 2:
12             imprima("Você digitou 2");
13             pare;
14         caso 3:
15             imprima("Você digitou 3");
16             pare;
17         default:
18             imprima("Você não digitou 1, nem 2, nem 3");
19     }
20 fim

```

O funcionamento é o seguinte: o valor da variável **x** é comparado a cada uma das constantes de cada **caso** (valores à frente da palavra **caso**) (1, 2 ou 3, neste exemplo). Se for igual, o bloco de comando após os dois pontos (**:**) é executado até o comando **pare**. Se não for igual a nenhuma das opções, o bloco correspondente à opção **default** é executado.

A escolha poderá ser feita **apenas** com números inteiros ou caracteres (uma letra, por exemplo) e o teste de cada caso (**case**) só testa a igualdade (não testa **>**, **<**, **>=**, etc, como pode ocorrer no **se (if)**).

Vamos a um **exercício**: escrever um algoritmo que leia um peso na Terra e o número de um planeta e imprima o valor deste peso no planeta escolhido. A relação de planetas é dada na Tabela 3.1, juntamente com os valores das gravidades relativas à da Terra.

Número	Gravidade relativa	Planeta
1	0.376	Mercúrio
2	0.903	Vênus
3	0.380	Marte
4	2.340	Júpiter
5	1.160	Saturno
6	1.150	Urano
7	1.190	Netuno

Tabela 3.1: Gravidades relativas em outros planetas

A seguir a solução, no algoritmo 3.3.

Algoritmo/Programa 3.3: Cálculo do peso em outros planetas

```
1 //alg011.txt
2
3 início
4     int planetaEscolhido;
5     float pesoNaTerra, gravidadeRelativa, pesoNoPlaneta;
6
7     escreva("Planetas");
8     escreva("1 - Mercúrio");
9     escreva("2 - Vênus");
10    escreva("3 - Marte");
11    escreva("4 - Júpiter");
12    escreva("5 - Saturno");
13    escreva("6 - Urano");
14    escreva("7 - Netuno");
15    escreva("Digite o número do planeta escolhido:");
16    ler(planetaEscolhido);
17
18    escreva("Informe seu peso na Terra:");
19    ler(pesoNaTerra);
20
21    escolha (planetaEscolhido) {
22        caso 1:
23            gravidadeRelativa = 0.376;
24            pare;
25        caso 2:
26            gravidadeRelativa = 0.903;
27            pare;
```

```
28         caso 3:
29             gravidadeRelativa = 0.380;
30             pare;
31         caso 4:
32             gravidadeRelativa = 2.340;
33             pare;
34         caso 5:
35             gravidadeRelativa = 1.160;
36             pare;
37         caso 6:
38             gravidadeRelativa = 1.150;
39             pare;
40         caso 7:
41             gravidadeRelativa = 1.190;
42             pare;
43         default:
44             gravidadeRelativa = 0;
45     }
46
47     se ( gravidadeRelativa != 0 ) {
48         pesoNoPlaneta = pesoNaTerra * gravidadeRelativa;
49         escreva("Seu peso no planeta escolhido é ", pesoNoPlaneta);
50     }
51     senão {
52         escreva("Erro na escolha do planeta.");
53     }
54
55 fim
```

Em C, o comando **escolha-caso** corresponde ao **switch-case** e o **pare** corresponde ao **break**.

3.2 switch-case: aplicações em C

A sintaxe da estrutura **switch-case** em C está no trecho de programa 3.4.

Algoritmo/Programa 3.4: O comando **switch-case**

```
1 //prog018.c
2
3 switch (variável) {
4     case valor_1:
5         comando_1;
6         comando_2;
7         . . .
8         comando_n;
9     break;
```

```
10     case valor_2:
11         comando_n+1;
12         comando_n+2;
13         . . .
14         comando_n+k;
15         break;
16     case valor_3:
17         comando_n+k+1;
18         comando_n+k+2;
19         . . .
20         comando_n+k+m;
21         break;
22     . . .
23     . . .
24     default:
25         comando_n+k+m+y+1;
26         comando_n+k+m+y+2;
27         . . .
28         comando_n+k+m+y+x;
29 }
```

3.3 Exercícios Propostos

3.3.1 Peso nos planetas

Implementar em C o algoritmo 3.3.

Resposta: programa 9.9.

3.3.2 Questão de múltipla escolha

Fazer um programa que mostre uma questão de múltipla escolha com 5 opções (letras **a**, **b**, **c**, **d**, e **e**). Sabendo a resposta certa, receber a opção do usuário e informar a letra que o usuário marcou e se a resposta está certa ou errada. Usar o tipo **char** para armazenar a variável de teste do **switch**. Ela pode ser lida do teclado com **scanf** e a máscara **%c** ou com a função **getchar()** (**opcao = getchar()**). Na comparação do **case**, deve-se colocar o valor a ser comparado entre aspas simples: **case 'a'**, por exemplo.

Resposta: programa 9.10.

3.3.3 Calculadora simples

Fazer um programa que lê dois números, lê a operação desejada (+ - * /), faz a operação pedida e mostra o resultado. A operação escolhida deve ser armazenada em uma variável do tipo **char**.

Resposta: programa 9.11.

Capítulo 4

For (para)

4.1 Descrição e exemplos

É comum termos um trecho de código (conjunto de instruções) que precise ser repetido várias vezes (executado várias vezes), até que uma condição seja satisfeita (ou até que uma condição deixe de ser satisfeita). Uma das formas disto ser feito é através da estrutura **para** (**for** em C). Material desenvolvido por Philippe Jabour - <http://www.jabour.com.br>.

A forma geral do comando **para** (chamado de **laço**) é a seguinte:

```
para ( inicialização ; condição ; incremento ) {  
    comando_1;  
    comando_2;  
    . . .  
    comando_n;  
}
```

A **inicialização** é geralmente uma atribuição, ou seja, uma variável recebe um valor inicial. Esta variável é chamada **variável de controle do laço**. A **condição** é um teste lógico que verifica se o laço será executado. É sempre testada no início e, somente se for verdadeira, o interior do laço é executado. O **incremento** define como a variável de controle do laço varia cada vez que o laço é repetido.

Vejamos um exemplo no algoritmo 4.1.

Algoritmo/Programa 4.1: Um exemplo simples com o laço **para**

```
1 //alg012.txt  
2  
3 início  
4     int i;
```

```

5
6     para ( i = 0 ; i < 100 ; i = i + 1 ) {
7         imprima ( i );
8     }
9
10 fim

```

A variável **i** recebe inicialmente o valor **0**. O teste lógico (condição de permanência) $i < 100$ é feito e retorna verdadeiro. Deste modo, o laço é executado e o valor **0** é impresso. Ao final do laço, a variável **i** é incrementada em uma unidade e passa a valer **1** ($i = i + 1$). A condição continua verdadeira pois $1 < 100$. Assim, o laço é executado novamente e o valor **1** é impresso. Isto se repete até que **i** assumo o valor **100**. Com $i = 100$, a condição se torna falsa e o laço é abandonado (o algoritmo passa para a próxima instrução após a chave de fechamento do bloco). Em resumo, o algoritmo imprime todos os números inteiros de 0 a 99 (inclusive).

Um exercício: Faça um algoritmo para imprimir os números pares maiores que zero e menores que 1000.

Solução no algoritmo 4.2.

Algoritmo/Programa 4.2: Imprimir os números pares de 0 a 1000

```

1 início
2     int par;
3
4     para ( par = 2 ; par < 1000 ; par = par + 2 ) {
5         imprima ( par );
6     }
7
8 fim

```

Vamos agora analisar o algoritmo 4.3:

Algoritmo/Programa 4.3: **para** com decremento da variável

```

1 início
2     int x, y;
3     float z;
4
5     para ( x = 100 ; x != 65 ; x = x - 5 ) {
6         z = x * x;
7         escreva ("O quadrado de %d é %f", x, z);
8     }
9 fim

```

Inicialmente, as variáveis são declaradas, como sempre. x é a variável de controle do laço **para** e é inicializada com o valor 100. O primeiro teste é atendido (dá verdadeiro já que $x \neq 65$). Deste modo, o algoritmo “entra no para”, calcula o quadrado de x ($x \times x$) e guarda o resultado na variável z . Em seguida, os valores de x e z são impressos e termina o laço. Neste momento, x é **decrementado** em 5 unidades e passa a valer 95. A condição

de permanência continua sendo atendida e o laço é executado novamente. Isto ocorre até que x assumo o valor 65. Neste momento $x \neq 65$ se torna falso e o laço é abandonado.

O algoritmo 4.3 implementado em C encontra-se no programa 4.4 e a saída (resultado) do programa na Tabela 4.1.

Algoritmo/Programa 4.4: Algoritmo 4.3 implementado em C

```
1 //prog021.c
2
3 #include <stdio.h>
4
5 int main() {
6     int x, y;
7     float z;
8
9     for ( x = 100 ; x != 65 ; x = x - 5 ) {
10         z = x * x;
11         printf("\n0 quadrado de %d é %.0f", x, z);
12     }
13 }
```

O quadrado de 100 é 10000
O quadrado de 95 é 9025
O quadrado de 90 é 8100
O quadrado de 85 é 7225
O quadrado de 80 é 6400
O quadrado de 75 é 5625
O quadrado de 70 é 4900

Tabela 4.1: Saída do programa 4.4

Considere o trecho de programa abaixo:

```
x = 10;
para ( y = 10 ; y != x ; y = y + 1 ) {
    printf("\n y = %d",y);
}
```

Observe que o interior do laço nunca será executado. Como $x = y$, o teste $x \neq y$ já dará **falso** da primeira vez.

4.2 for: aplicações em C

A sintaxe do **for**:

```

for ( inicialização ; condição ; incremento ) {
    comando_1;
    comando_2;
    . . .
    comando_n;
}

```

Observação: é comum o uso de $y++$ no lugar de $y = y + 1$. Da mesma forma, $x--$ é a mesma coisa que $x = x - 1$.

Vamos a um exemplo: imprimir os números ímpares existentes entre 30 e 100. A solução está no programa 4.5:

Algoritmo/Programa 4.5: Imprimir os ímpares de 30 a 100

```

1 //prog022.c
2
3 #include <stdio.h>
4
5 int main() {
6     int impar;
7
8     printf("\nÍmpares entre 30 e 100: ");
9
10    for ( impar = 31 ; impar < 100 ; impar = impar + 2 ) {
11        printf("%d, ", impar);
12    }
13    printf("\n");
14 }

```

Bastou identificar que o primeiro ímpar era o 31, inicializar a variável de controle com este valor e rodar o **for** de 2 em 2, até o limite de 100.

Em C, $x = x + 2$ é a mesma coisa que $x += 2$; $y = y - 5$ é a mesma coisa que $y -= 5$; $z = z * 8$ é a mesma coisa que $z *= 8$; $k = k/9$ é a mesma coisa que $k /= 9$; e assim por diante.

Na opção **incremento**, a terceira do comando **for**, podemos ter operações com mais de uma variável. Observe no programa 4.6 que as três variáveis i , x e y são modificadas a cada rodada do laço (cada rodada do **for** é também chamada de iteração).

Algoritmo/Programa 4.6: 3 variáveis mudando a cada iteração

```

1 //prog023.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     int i, x = 2, y = 20;
8
9     for ( i = 1 ; i < 10 ; i++, x *= 2, y -= 1 ) {
10         printf("\n i = %d \t x = %d \t y = %d ", i, x, y);    // \t corresponde a um TAB
11             (tabulação)
12     }
13     printf("\n");
14
15 }

```

Podemos ter **aninhamento** de laços **for**, ou seja, um **for** dentro do outro. Veja o programa 4.7:

Algoritmo/Programa 4.7: Um **for** dentro do outro

```

1 //prog026.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     int i, j;
8
9     printf("\n Impressão de (i,j)\n");
10
11     for ( i = 0 ; i < 10 ; i++ ) {
12         printf("\n");
13         for ( j = 0 ; j < 10 ; j++ )
14             printf("\t(%d, %d)", i, j);
15     }
16
17     printf("\n\n");
18
19 }

```

A saída em vídeo será a seguinte:

```

(0, 0) (0, 1) (0, 2) (0, 3) (0, 4) (0, 5) (0, 6) (0, 7) (0, 8) (0, 9)
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4) (1, 5) (1, 6) (1, 7) (1, 8) (1, 9)
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4) (2, 5) (2, 6) (2, 7) (2, 8) (2, 9)
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4) (3, 5) (3, 6) (3, 7) (3, 8) (3, 9)
(4, 0) (4, 1) (4, 2) (4, 3) (4, 4) (4, 5) (4, 6) (4, 7) (4, 8) (4, 9)

```

(5, 0) (5, 1) (5, 2) (5, 3) (5, 4) (5, 5) (5, 6) (5, 7) (5, 8) (5, 9)
(6, 0) (6, 1) (6, 2) (6, 3) (6, 4) (6, 5) (6, 6) (6, 7) (6, 8) (6, 9)
(7, 0) (7, 1) (7, 2) (7, 3) (7, 4) (7, 5) (7, 6) (7, 7) (7, 8) (7, 9)
(8, 0) (8, 1) (8, 2) (8, 3) (8, 4) (8, 5) (8, 6) (8, 7) (8, 8) (8, 9)
(9, 0) (9, 1) (9, 2) (9, 3) (9, 4) (9, 5) (9, 6) (9, 7) (9, 8) (9, 9)

4.3 Exercícios Propostos

4.3.1 Adivinhar um número com n tentativas

Modifique o exercício 2.3.1 de modo que o usuário informe quantas tentativas de adivinhação quer fazer. O programa permite então que ele tente este número de vezes informado ou acaba caso ele acerte antes.

Obs.: O comando **break** interrompe o laço **for**.

Resposta: programa 9.12.

4.3.2 Menu de opções e um for

Fazer um algoritmo que tenha um menu que permita ao usuário escolher 5 opções de exibição de potências. As opções permitem escolher as potências de 2, 3, 5, 10 ou 15. De acordo com a opção escolhida, o programa exibirá todas as potências do número escolhido que sejam menores que 10000.

Resposta: programa 9.13.

4.3.3 Progressão aritmética

Considere uma progressão aritmética (PA). Faça um programa que receba o termo inicial a_1 , a razão r e o número de termos a serem gerados. Em seguida, a PA é impressa, com 10 termos por linha. À medida que a PA vai sendo gerada, a soma dos termos é calculada (vai sendo acumulada). Ao final, esta soma é impressa e é feita uma verificação de acerto através da fórmula da soma dos termos da PA dada pela expressão 4.1.

$$S_n = \frac{n \times (a_1 + a_n)}{2} \quad (4.1)$$

Resposta: programa 9.14.

4.3.4 Sequência de Fibonacci

Faça um programa que receba um número n e gere os n primeiros números da sequência de Fibonacci definida na fórmula 4.2.

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases} \quad (4.2)$$

Resposta: programa 9.15.

4.3.5 Par de números com soma definida

Faça um programa que leia um valor inteiro não negativo e imprima todos os pares de números inteiros não negativos cuja soma seja igual a este número. Considere que o par x, y é diferente de y, x e deve ser impresso duas vezes.

Dica que pode ser usada em uma solução: a inicialização do **for** (primeira opção entre parênteses), pode conter mais de uma variável. Por exemplo: **for (x = 0, y = 100; etc. . .)**.

Resposta: programa 9.16.

4.3.6 Soma de alguns números digitados

Faça um programa que receba 10 números via teclado e imprima a soma dos números cujos quadrados sejam menores que 225.

Resposta: programa 9.17.

4.3.7 Achar os divisores de um número

Ler um número e imprimir todos os seus divisores.

Resposta: programa 9.18.

4.3.8 Imprimir somente acima da diagonal principal

Considere a saída do programa 4.7 como uma matriz. Modifique este programa para que apenas a diagonal principal e os dados acima dela sejam impressos.

Resposta: programa 9.19.

Capítulo 5

while e do-while (enquanto e faça-enquanto)

5.1 Descrição e exemplos

Neste capítulo vamos estudar dois outros comandos de repetição. O primeiro deles é o **enquanto** e sua forma geral é a seguinte:

```
enquanto ( condição ) {  
    comando_1;  
    comando_2;  
    . . .  
    comando_n;  
}
```

A **condição** é um teste lógico que verifica se o laço será executado. É sempre testada no início e, somente se for verdadeira, o interior do laço é executado. Cada vez que o **enquanto** é executado, ao se atingir o fim do laço (**}**), a execução volta ao início (**enquanto (condição) {**), a condição é testada novamente. Se for verdadeira, o laço é executado novamente, se for falsa, a execução segue para a próxima instrução após o laço.

O algoritmo 5.1 apresenta um primeiro exemplo.

Algoritmo/Programa 5.1: Um exemplo simples com o laço **enquanto**

```
1 início  
2  
3     int par = 2;  
4
```

```
5     enquanto ( par <= 1000 ) {
6         escreva ( par );
7         par = par + 2;
8     }
9
10 fim
```

A variável **par** recebe inicialmente o valor **2**. O teste lógico (**condição de permanência**) $par \leq 1000$ é feito e retorna verdadeiro. Deste modo, o laço é executado e o valor **2** é impresso. Ao final do laço, a variável **par** é incrementada em duas unidades e passa a valer **4** ($par = par + 2$). A condição continua verdadeira pois $4 \leq 1000$. Assim, o laço é executado novamente e o valor **4** é impresso. Isto se repete até que **par** assumo o valor 1002. Com $par = 1002$, a condição se torna falsa e o laço é abandonado (o algoritmo passa para a próxima instrução após a chave de fechamento do bloco). Em resumo, o algoritmo imprime todos os números pares de 2 (inclusive) a 1000 (inclusive).

O outro comando é o **faça-enquanto**. Ele é semelhante ao **enquanto**, mas o teste lógico (**condição**) é feito no fim do bloco. Sua forma geral é a seguinte:

```
faça {
    comando_1;
    comando_2;
    . . .
    comando_n;
} enquanto ( condição );
```

Observe que a condição fica depois das chaves de fechamento do bloco, que o bloco é executado **pelo menos uma vez** e que se usa o **ponto e vírgula** ao final da condição.

Vejam os então, no Algoritmo 5.2, como fica o Algoritmo 5.1 (pares de 2 a 1000) desenvolvido com o **faça-enquanto**:

Algoritmo/Programa 5.2: Um exemplo simples com o laço **faça-enquanto**

```
1  inicio
2
3      int par = 2;
4
5      faça {
6          escreva ( par );
7          par = par + 2;
8      } enquanto ( par <= 1000 );
9
10 fim
```

Observe que o 2 é impresso inicialmente sem que qualquer condição seja testada (**entra do laço sempre**). Após a impressão do 2, o valor da variável **par** é incrementado em duas unidades e o teste é feito ($4 \leq 1000$?). Este teste retorna verdadeiro até **par** valer 1000. Neste momento, $1000 \leq 1000$ ainda retorna **verdadeiro**, o laço é executado, o valor 1000 é impresso e **par** passa a valer 1002. Como $1002 \leq 1000$ retorna **falso**, o laço é **abandonado** e o fluxo de execução segue para a próxima instrução **após o faça-enquanto**.

5.2 while e do-while: aplicações em C

Em C, o **enquanto** é substituído por **while** e o **faça-enquanto** por **do-while**. Veja a sintaxe:

```
while ( condição ) {  
    comando_1;  
    comando_2;  
    . . .  
    comando_n;  
}
```

```
do {  
    comando_1;  
    comando_2;  
    . . .  
    comando_n;  
} while ( condição );
```

Como nos comandos anteriores, podemos ter aninhamento, com **while** dentro de **while**, **do-while** dentro de **do-while**, **while** dentro do **switch**, **do-while** dentro do **for**, etc, etc, etc.

Podemos usar **while** ou **do-while** para repetir um menu até que o usuário escolha sair. Veja no exemplo 5.3.

Algoritmo/Programa 5.3: Uso do **while** combinado com um menu de opções

```
1 //prog035.c  
2  
3 #include <stdio.h>  
4  
5 int main() {  
6  
7     int opcaoDoMenu = 0;  
8  
9     while ( opcaoDoMenu != 5 ) {  
10  
11         printf("\n Opções ");
```

```
12     printf("\n 1 - Faz isso");
13     printf("\n 2 - Faz aquilo");
14     printf("\n 3 - Calcula alguma coisa");
15     printf("\n 4 - Mostra aquele resultado");
16     printf("\n 5 - Sai do programa\n -> ");
17     scanf("%d", &opcaoDoMenu);
18
19     switch(opcaoDoMenu) {
20
21         case 1:
22             printf("\nVocê escolheu a opção 1.");
23             printf("\nA opção se refere a \"Fazer isso\".");
24             printf("\nAqui entraria o código que faria o que se deseja.\n\n");
25             break;
26
27         case 2:
28             printf("\nVocê escolheu a opção 2.");
29             printf("\nA opção se refere a \"Fazer aquilo\".");
30             printf("\nAqui entraria o código que faria o que se deseja.\n\n");
31             break;
32
33         case 3:
34             printf("\nVocê escolheu a opção 3.");
35             printf("\nA opção se refere a \"Calcular alguma coisa\".");
36             printf("\nAqui entraria o código que faria o que se deseja.\n\n");
37             break;
38
39         case 4:
40             printf("\nVocê escolheu a opção 4.");
41             printf("\nA opção se refere a \"Mostrar aquele resultado\".");
42             printf("\nAqui entraria o código que faria o que se deseja.\n\n");
43             break;
44
45         case 5:
46             printf("\nVocê escolheu a opção 5.");
47             printf("\nA opção se refere a \"Sair do programa\".\n\n");
48             break;
49
50         default:
51             printf("\nVocê escolheu uma opção inválida.\n\n");
52
53     }
54
55 }
```

56 }

Material desenvolvido por Philippe Jabour - <http://www.jabour.com.br>.

5.3 Exercícios Propostos

5.3.1 Adivinhar n° até o usuário acertar ou desistir

Modifique os exercícios 2.3.1 e 4.3.1 de modo que o usuário possa tentar adivinhar o número quantas vezes desejar, tendo a opção de abandonar o programa quando quiser. Aumente agora o intervalo de números gerados para $[0, 99]$.

Resposta: programa 9.20.

5.3.2 Preço de passagem aérea

Fazer um programa com o seguinte menu:

1. Informa preço da passagem
2. Informa quantos preços já foram informados
3. Informa total em R\$ já informados
4. Sai do programa.

Para a opção 1, o programa recebe o destino e se é viagem só de ida ou de ida e volta e informa o preço, segundo a tabela abaixo:

Destino	Preço ida	Preço ida e volta
Região Norte	R\$ 500	R\$ 900
Região Nordeste	R\$ 350	R\$ 650
Região Centro-oeste	R\$ 350	R\$ 600
Região Sul	R\$ 300	R\$ 550

Tabela 5.1: Preço de passagens aéreas por região

5.3.3 Lê n°s e exhibe o maior, o menor e a quantidade

Fazer um programa que leia “n” números e imprima, no final, o maior deles, o menor deles e quantos números foram digitados. A opção para terminar de informar números deve ser -999.

5.3.4 Menu para somar ou multiplicar n°s

Fazer um programa com o seguinte menu:

- a. Soma vários números
- b. Multiplica vários números
- c. Sai do programa

Cada opção do menu faz o que o nome sugere.

5.3.5 Achar o número de múltiplos em um intervalo

Fazer um programa que leia X e N e informe quantos múltiplos de X existem entre 1 e N . Use **while** ou **do-while** e não use divisão.

Exemplo: Se $X = 8$ e $N = 30$, a saída do programa será 3. Pois existem 3 múltiplos de 8 entre 1 e 30 (8, 16 e 24).

Capítulo 6

String, vetor e matriz

6.1 Vetor

Variáveis armazenam **um** valor de um determinado tipo. Por exemplo, uma variável do tipo **int** pode armazenar um valor do tipo inteiro (**int**).

Por exemplo:

int a; declara a variável **a**. Esta variável pode assumir o valor 10 **ou** -5 **ou** 8, etc.

Um **vetor** (ou **array**), pode armazenar **vários** valores de um determinado tipo. É como ter vários valores do mesmo tipo colocados um do lado do outro e todos acessados pela variável de mesmo nome.

Para acessar cada um destes valores, usamos um **índice** numérico que determina qual deles queremos manipular.

A sintaxe é a seguinte:

tipo nome[tamanho];

Por exemplo:

int notas[5];

Neste exemplo queremos armazenar 5 valores do tipo **int** na variável **notas**.

Em C, vetores começam na posição 0. Os elementos do vetor ocupam localizações adjacentes na memória.

Deste modo, para o exemplo acima, teremos as seguintes posições válidas para o vetor **notas**:

notas[0], notas[1], notas[2], notas[3], notas[4].

Vejam os primeiros exemplos no programa 6.1.

Algoritmo/Programa 6.1: O primeiro exemplo com **vetores**

```
1 //prog036.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     int x;
8
9     int vetorTeste[5];
10
11     vetorTeste[0] = 34;
12     vetorTeste[1] = -3;
13     vetorTeste[2] = 0;
14     vetorTeste[3] = 6;
15     vetorTeste[4] = -8;
16
17     printf("Elementos do vetor:");
18     printf("\nElemento da posição 0 do vetor = %d", vetorTeste[0]);
19     printf("\nElemento da posição 1 do vetor = %d", vetorTeste[1]);
20     printf("\nElemento da posição 2 do vetor = %d", vetorTeste[2]);
21     printf("\nElemento da posição 3 do vetor = %d", vetorTeste[3]);
22     printf("\nElemento da posição 4 do vetor = %d\n", vetorTeste[4]);
23
24     x = vetorTeste[3] * vetorTeste[4];
25     printf("\nElemento 3 multiplicado pelo elemento 4 = %d\n", x);
26
27     vetorTeste[0] *= 2;
28     printf("\nElemento da posição 0 multiplicado por 2 = %d\n", vetorTeste[0]);
29
30     printf("\nDigite um novo valor para o elemento 4:");
31     scanf("%d",&vetorTeste[4]);
32     printf("\nNovo valor do elemento da posição 4 do vetor = %d\n\n", vetorTeste[4]);
33
34 }
```

Importante: C não verifica se o programa está acessando uma posição válida do vetor. Se você acessar uma posição além do fim do vetor, erros em tempo de execução podem ocorrer. Veja abaixo um trecho de código que pode provocar erros imprevisíveis:

```
int vetorDois[10];
int x;
vetorDois[11] = 32;
x = vetorDois[13];
```

Podemos ter variáveis usadas como índice do vetor. Vejamos o programa 6.2:

Algoritmo/Programa 6.2: Variável como índice do vetor

```
1 //prog037.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     int i;    //tem que ser do tipo inteiro
8
9     int vetor[10];    //declara um vetor de inteiros
10
11     //Insere o quadrado dos números de 0 a 9 em cada posição do vetor
12     for ( i = 0 ; i < 10 ; i ++ ) {
13         vetor[i] = i * i;
14     }
15
16     //Exibe o conteúdo do vetor
17     for ( i = 0 ; i < 10 ; i ++ ) {
18         printf("\n%d", vetor[i]);
19     }
20
21     printf("\n");
22
23 }
```

O índice (número entre colchetes), tem que ser do tipo **int**.

6.2 String

Uma **string** é uma cadeia ou sequência de caracteres. As **strings** são usadas para armazenar nomes, palavras e frases.

Em C, uma **string** consiste em um vetor de caracteres (**char**) terminada em zero. Mais especificamente, terminada com `\0`. Por essa razão, você deve declarar o vetor de caracteres com uma posição a mais do que a maior **string** que você quer que ele possa armazenar. Por exemplo, se você quer que o vetor armazene uma **string** de até 10 posições, declare da seguinte maneira:

```
char palavra[11];
```

A melhor maneira de ler uma **string** via teclado é com a função **gets()**:

```
gets(nome_do_vetor);
```

Por exemplo:

```
gets(palavra);
```

Veja um exemplo simples no programa 6.3:

Algoritmo/Programa 6.3: Um exemplo simples de string e gets()

```
1 //prog039.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     char frase[80];
8
9     printf("Digite uma frase: ");
10
11    gets(frase);
12
13    printf("Você digitou %s\n",frase);
14
15 }
```

A função `gets()` lerá os caracteres digitados até que você tecle **ENTER**. Não é feita nenhuma verificação se o que você digitou cabe no vetor declarado. No programa 6.3, se você digitar uma **string** com mais de 79 caracteres, os últimos caracteres invadirão espaços de memória de outras variáveis.

6.2.1 Funções de manipulação de strings

Existem algumas funções prontas para a manipulação de **strings**. Vamos descrevê-las e apresentar alguns exemplos. Estas funções estão da biblioteca **string.h**, sendo necessário inserir `#include <string.h>` no cabeçalho do programa.

strlen()

Sintaxe: `strlen(s)` (onde `s` é uma **string**).

Esta função retorna (devolve) um número inteiro que corresponde ao comprimento (tamanho ou número de caracteres) da **string s**.

Como ela retorna um valor, é comum fazermos uma variável inteira receber o resultado da função.

Veja o exemplo no programa 6.4.

Algoritmo/Programa 6.4: A função `strlen()`

```
1 //prog041.c
2
3 #include <stdio.h>
4 #include <string.h>
5
6 int main() {
7
8     int tamanho;
9
10    char s[30];
11
12    printf("\nDigite o conteúdo da string: ");
13
14    gets(s);
15
16    tamanho = strlen(s);
17
18    printf("\nA string\t%s\ttem %d caracteres.\n",s,tamanho);
19
20 }
```

Observe que, como a função `strlen()` retorna um valor inteiro, ao fazermos `tamanho = strlen(s);`, este valor é atribuído à variável `tamanho`.

strcpy()

Sintaxe: `strcpy(s1, s2)` (onde `s1` e `s2` são strings).

Esta função copia o conteúdo da string `s2` para a string `s1`. Certifique-se que $s2 \leq s1$.

Veja o exemplo no programa 6.5.

Algoritmo/Programa 6.5: A função `strcpy()`

```
1 //prog042.c
2
3 #include <stdio.h>
4 #include <string.h>
5
6 int main() {
7
8     char teste[50];
9
10    strcpy(teste, "Texto a ser inserido na string teste");
11
12    printf("\n %s \n",teste);
```

```
13  
14 }
```

strcat()

Sintaxe: **strcat(s1, s2)** (onde **s1** e **s2** são **strings**).

Esta função anexa **s2** ao final de **s1**. **s2** fica inalterada. Dizemos que foi feita a concatenação das strings **s1** e **s2**

O programa 6.6 apresenta um exemplo.

Algoritmo/Programa 6.6: A função strcat()

```
1 //prog043.c  
2  
3 #include <stdio.h>  
4 #include <string.h>  
5  
6 int main() {  
7  
8     char primeiraString[50], segundaString[50];  
9  
10    printf("\nDigite a primeira string.\n");  
11    gets(primeiraString);  
12    printf("\nDigite a segunda string.\n");  
13    gets(segundaString);  
14  
15    printf("\n\nAntes da concatenação:");  
16  
17    printf("\nPrimeira: %s", primeiraString);  
18    printf("\nSegunda: %s", segundaString);  
19  
20    strcat(primeiraString, segundaString);  
21  
22    printf("\n\nDepois da concatenação:");  
23  
24    printf("\nPrimeira: %s", primeiraString);  
25    printf("\nSegunda: %s\n\n", segundaString);  
26  
27 }
```

strcmp()

Sintaxe: **strcmp(s1, s2)** (onde **s1** e **s2** são **strings**).

Esta função **compara** as strings **s1** e **s2**. Ela retorna 0 se $s1 == s2$; retorna um número positivo se **s1** **for lexicograficamente maior que s2** e retorna um número negativo se $s1 < s2$.

Algoritmo/Programa 6.7: A função strcmp()

```
1 //prog040.c
2
3 #include <stdio.h>
4 #include <string.h>
5
6 int main() {
7
8     int resultado;
9
10    char s1[10], s2[10];
11    strcpy(s1, "aaa");
12    strcpy(s2, "aaa");
13    resultado = strcmp(s1, s2);
14    printf("\nResultado = %d", resultado);
15
16    strcpy(s1, "aaa");
17    strcpy(s2, "bbb");
18    resultado = strcmp(s1, s2);
19    printf("\nResultado = %d", resultado);
20
21    strcpy(s1, "bbb");
22    strcpy(s2, "aaa");
23    resultado = strcmp(s1, s2);
24    printf("\nResultado = %d\n", resultado);
25
26    strcpy(s1, "aaaa");
27    strcpy(s2, "aaa");
28    resultado = strcmp(s1, s2);
29    printf("\nResultado = %d\n", resultado);
30
31 }
```

6.3 Matriz

Matrizes são vetores com mais dimensões. Em outras palavras, podemos ver um vetor como uma matriz com uma linha e n colunas. Uma matriz bidimensional possui i linhas e j colunas. C permite matrizes com mais dimensões: 3, 4, ... (multidimensionais).

No momento, vamos abordar apenas as matrizes bidimensionais.

A sintaxe é a seguinte:

tipo nome[dimensão_1] [dimensão_2];

Por exemplo:

int matriz[10][20]; declara uma matriz de inteiros com 10 linhas e 20 colunas.

Outros exemplos:

int matrizA[5][5]; : a variável chama-se **matrizA**, seus elementos são do tipo **int** e sua dimensão é de 5 linhas e 5 colunas.

matrizA[2][2] = **34**; : a atribuição de valores é normal, como se faz com qualquer variável ou vetor.

float M[4][6] : a variável chama-se **M**, seus elementos são do tipo **float** e sua dimensão é de 4 linhas e 6 colunas.

M[2][3] refere-se ao elemento da terceira linha e quarta coluna da matriz **M**. Como nos vetores, a contagem começa de **zero**.

M[i][j] = **3.456**; : podemos usar variáveis como índices da matriz.

De um modo geral, temos **tipo nome_da_variável**[nº de linhas][nº de colunas].

Vejamos um exemplo no programa 6.8.

Algoritmo/Programa 6.8: Um exemplo simples com matriz

```

1 //prog044.c
2
3 #include <stdio.h>
4
5 int main(){
6
7     int i, j, matriz[3][4];
8
9     printf("\n Digite os elementos da matriz\n");
10
11     for ( i = 0 ; i < 3 ; i++ ) {
12         for ( j = 0 ; j < 4 ; j++ ) {
13             scanf("%d",&matriz[i][j]);
14         }
15     }
16
17     for ( i = 0 ; i < 3 ; i++ ) {
18         for ( j = 0 ; j < 4 ; j++ ) {
19             printf("matriz [%d] [%d] = %d\t", i, j, matriz[i][j]);
20         }
21         printf("\n");
22     }

```

23 }

6.3.1 Matrizes de strings

Para criar uma matriz de **strings**, usa-se uma matriz bidimensional de caracteres (**char**), na qual o índice da esquerda (**linhas**) determina o número de **strings** e o índice da direita (**colunas**) especifica o comprimento máximo de cada **string**.

Por exemplo, **char matriz_str[30][81]**; declara uma matriz de 30 **strings**, sendo que cada **string** pode ter, no máximo, 81 caracteres (80 caracteres + o terminador `\0`).

Para acessar uma determinada **string**, basta especificar o índice da linha (primeiro índice).

A linha abaixo, por exemplo, lê a terceira **string** da **matriz_str**:

```
gets(matriz_str[2]);
```

Ela é equivalente a **gets(matriz_str[2][0])**;

Vamos a um exemplo que lê várias linhas de texto.

Algoritmo/Programa 6.9: Lê e exibe um texto com várias linhas

```

1 //prog045.c
2
3 #include <stdio.h>
4
5 int main(){
6
7     int i, j;
8     char texto[100][80];
9
10    printf("-----");
11    printf("\n Digite o texto e uma linha em branco para encerrar\n");
12    printf("-----\n");
13
14    for ( i = 0 ; i < 100 ; i++ ) {
15
16        gets(texto[i]);
17
18        if ( texto[i][0] == '\0' )
19            break;
20    }
21
22    printf("-----\n");
23
24    for ( i = 0 ; i < 100 ; i++ ) {
```

```
25
26         if ( texto[i][0] == '\0' )
27             break;
28
29         printf("%s\n", texto[i]);
30
31     }
32
33     printf("-----\n");
34
35 }
```

O primeiro **for** recebe (lê) várias linhas digitadas com a função **gets()**. Quando uma linha em branco é digitada, ela terá apenas uma **string** vazia, a condição do **if** será verdadeira e o **for** será interrompido com o comando **break**.

Em seguida, todas as **strings** (linhas do texto) são impressas até atingirmos a linha vazia (**texto[i][0] == '\0'**).

6.4 Exercícios Propostos

6.4.1 Calcular a média dos elementos de um vetor

Faça um programa que receba via teclado 8 notas de uma turma. **Depois** de lidas todas as notas, calcule e imprima a média da turma. As notas devem ser armazenadas em um vetor de reais (**float**).

Resposta: programa 9.21.

6.4.2 Gerar vetor soma e produto a partir de 2 vetores originais

Ler 2 vetores de inteiros com 11 elementos cada um. Em seguida, gerar um vetor soma onde cada elemento corresponda à soma dos dois elementos correspondentes dos vetores lidos. Gerar outro vetor produto cujos elementos correspondam ao produto dos elementos de mesma posição dos vetores lidos. Exibir os 4 vetores na tela.

6.4.3 Imprimir vetor de trás pra frente

Ler um vetor com 8 elementos do tipo **float** e imprimir seus elementos de trás pra frente (do último ao primeiro).

6.4.4 Concatenar 2 vetores

Ler 2 vetores com 7 elementos do tipo inteiro cada um e concatená-los em um terceiro vetor.

6.4.5 Achar maior e menor elementos do vetor e somar seus elementos

Fazer um programa que receba 15 valores inteiros e os grave em um vetor. Depois desta etapa, percorra o vetor e identifique o maior e o menor elemento do vetor. Informe os valores e a posição que eles ocupam no vetor. Depois desta etapa, some os elementos do vetor e exiba o valor desta soma.

6.4.6 Confere senha digitada

Fazer um programa que leia a senha do usuário e informe se a senha está correta ou incorreta. Considere que a senha correta está gravada no código fonte do programa.

6.4.7 Imprimir uma string de trás para frente

Fazer um programa que leia uma string e a imprima de trás pra frente.

Exemplo:

Digitado: ifsudestemg

Impresso: gmetseudsi

6.4.8 Lê e imprime strings e números

Faça um programa que receba o nome do aluno, o nome da disciplina, o número de faltas (inteiro) e a nota na disciplina (real). Em seguida, o programa imprime os dados lidos.

6.4.9 Soma de matrizes

Fazer um programa que preencha automaticamente uma matriz 3×3 com $0, 1, 2, 3, 4, \dots$. Em seguida, o programa recebe, via teclado, os elementos de outra matriz 3×3 . Por fim, o programa calcula e exibe a matriz soma das duas primeiras matrizes.

6.4.10 Soma dos elementos da diagonal principal

Fazer um programa que leia os elementos de uma matriz 4×4 . Em seguida, o programa calcula e exibe a soma dos elementos da diagonal principal.

6.4.11 Menu e matriz

Fazer um programa c/ seguinte menu:

1. Inserir notas
2. Exibir nota do aluno
3. Modifica nota
4. Média do aluno
5. Média da turma na disciplina
6. Média geral da turma
7. Exibir todas as notas
8. Sair

A turma tem 5 alunos e cada aluno tem 3 disciplinas. As notas devem ser inseridas em uma matriz 5×3 . Considere que as opções 2, 3, 4, 5, 6 e 7 só podem ser executadas se a opção 1 já tiver sido executada.

Considere que a opção 1 só pode ser executada uma vez.

A opção 2 pede o número do aluno e exibe suas notas.

A opção 3 pede o número do aluno e da disciplina, lê a nova nota e substitui a antiga pela nova.

A opção 4 pede o número do aluno, calcula e exibe sua média.

A opção 5 pede o número da disciplina, calcula e exibe a média da turma na disciplina.

Capítulo 7

Funções

7.1 Funções em C

Funções são trechos de código que podem ser referenciados por um nome. Quando o nome de uma função aparece no código, dizemos que a função foi “chamada”. Quando uma **função** é chamada, ela “faz alguma coisa” e pode retornar (devolver) valores (resultado).

Quando a **função** não está pronta em uma biblioteca, ela precisa ser criada (implementada).

Forma geral:

```
tipo_que_retorna nome_da_função (lista de parâmetros) {  
    código da função  
}
```

Exemplo:

```
float calculaMedia (float a, float b) {  
    código da função  
    return(resultado);  
}
```

A **função** se chama **calculaMedia**. Ela **retorna** (devolve) um valor real (**float**). Ela recebe dois argumentos do tipo **float** (a e b). Após calcular a média, o comando **return(resultado);** é responsável por retornar a média calculada para o programa que chamou a função.

A seguir o primeiro exemplo.

Algoritmo/Programa 7.1: Programa com a função calculaMedia

```
1 //prog046.c
2
3 #include <stdio.h>
4
5 float calculaMedia(float , float);
6
7 int main(){
8
9     float a, b, m;
10
11     printf("\nDigite o primeiro número: ");
12     scanf("%f", &a);
13     printf("\nDigite o segundo número: ");
14     scanf("%f", &b);
15
16     m = calculaMedia(a, b);
17
18     printf("\nA média entre %.2f e %.2f é %.2f\n",a,b,m);
19
20 }
21
22 float calculaMedia(float argA, float argB) {
23     float mediaCalculada;
24
25     mediaCalculada = ( argA + argB ) / 2;
26
27     return ( mediaCalculada);
28
29 }
```

Vamos analisar o código.

Observe que o programa contém **duas funções**, a **função main** e a **função calculaMedia**. A execução começa pela **função main**.

Na **linha 14**, a variável *m* recebe o resultado da função **calculaMedia**. Neste instante, a execução do programa **salta** para a **função calculaMedia** e ela começa a ser executada. Veja que na **chamada da função**, dois argumentos foram passados, as variáveis *a* e *b*. Estes argumentos são recebidos na função através das variáveis *argA* e *argB*. A média é então calculada e **devolvida (retornada)** através do comando **return**. O comando **return** encerra a execução de qualquer função. Assim, neste instante, a execução volta para a **função main**, o resultado retornado (a média entre os argumentos passados) é atribuída à variável *m* e o programa continua a sua execução.

Por fim a explicação da **linha 3**. Durante a tradução do programa para a linguagem do computador

(**compilação**), o tradutor (**compilador**) irá deparar com o nome da função **calculaMedia**. Se não houvesse a **linha 3**, função não seria reconhecida e ocorreria um erro de compilação. Deste modo, a **linha 3** serve para apresentar ao compilador a **função calculaMedia**, ou seja, diz ao compilador que esta função existe, o tipo de valor que ela retorna e os argumentos que recebe. Chamamos esta linha de **protótipo da função**.

Outra opção é escrever a função implementada **antes** da **função main**.

Uma função pode ser chamada (usada) várias vezes. Como aplicação deste conceito, faça o seguinte: no programa 7.1, declare outras duas variáveis *c* e *d*, leia *c* e *d* com a função **scanf**, calcule e exiba a média entre *c* e *d* usando novamente a função **calculaMedia**.

Veja outra possibilidade do uso de funções:

```
x = ( calculaMedia(a,b) + calculaMedia(c,d) ) / 2;
```

Observe que o resultado da função é inserido diretamente em uma expressão e o valor retornado é usado para um novo cálculo.

Um **vetor ou matriz** podem ser passados como **argumentos** para funções. Por ainda não termos estudado **ponteiros** em C, será apresentada apenas uma forma de fazê-lo. Ela consiste em passar apenas o nome do vetor (ou matriz) na chamada da função e em declarar, na construção da função, um vetor ou matriz igual ao argumento que será passado na chamada. Vejamos o exemplo do programa 7.2.

Algoritmo/Programa 7.2: Passando uma matriz como argumento da função

```
1 //prog047.c
2
3 #include <stdio.h>
4
5 void imprimeMatriz( float argMatriz [4][4] );
6
7 int main() {
8
9     float matriz [4][4];
10
11     int i, j;
12
13     printf("\nDigite os elementos da matriz\n");
14     for ( i = 0 ; i < 4 ; i++ ) {
15         for ( j = 0 ; j < 4 ; j++ ) {
16             scanf("%f",&matriz[i][j]);
17         }
18     }
19
20     imprimeMatriz(matriz);
21
22 }
```

```
23
24 void imprimeMatriz(float argMatriz[4][4]) {
25
26     int i, j;
27
28     printf("\nImpressão da matriz\n");
29     for ( i = 0 ; i < 4 ; i++ ) {
30         for ( j = 0 ; j < 4 ; j++ ) {
31             printf("%.1f\t", argMatriz[i][j]);
32         }
33         printf("\n");
34     }
35 }
```

7.2 Regras de escopo

Variáveis declaradas dentro de uma função são chamadas **variáveis locais**, ou seja, elas só podem ser usadas dentro da função. Em outras palavras, elas “só existem” dentro da função. Isto serve também para qualquer bloco de código delimitado por chaves (`{...}`). Neste caso, dizemos que o **escopo** destas variáveis está limitado à função dentro da qual elas foram declaradas.

Quando o programa é executado, uma variável local é criada quando a execução do bloco é iniciada (`{`) e destruída quando a execução do bloco é encerrada (`}`).

No trecho a seguir, cada variável x só existe dentro de cada uma das funções. Uma não interfere na outra. Ambas podem ser chamar x ou poderiam ter nomes diferentes.

```
...
int funcaoUm(int a){
    int x;
    x = a * a;
    return(x);
}
...
int funcaoDois(int m){
    int x;
    x = m * m * m;
    return(x);
}
```

Ao contrário das variáveis locais, as **variáveis globais** são conhecidas em todo o programa e podem ser

usadas por qualquer parte do código. Sua visibilidade é total. Uma variável global é criada através de uma declaração feita **fora de qualquer função** (inclusive, fora da própria função `main`).

7.3 Exercícios Propostos

7.3.1 Calculadora usando funções

Fazer um programa que execute até que o usuário decida encerrar. Na execução, o programa recebe 2 números, o usuário escolhe a operação desejada (`+` `-` `*` `/`) e o programa efetua a operação desejada. Use chamadas de funções para executar as operações. Não permita divisão por zero.

7.3.2 Equação do segundo grau usando funções

Refaça o exercício 2.3.5 usando funções. As seguintes funções devem ser criadas e usadas:

float calculaDelta(float, float, float) : recebe os coeficientes da equação e retorna o valor de delta.

float calculaRaizUm(float, float, float) : recebe os coeficientes a e b da equação, recebe o valor de delta e retorna o valor de uma das raízes da equação.

float calculaRaizDois(float, float, float) : recebe os coeficientes a e b da equação, recebe o valor de delta e retorna o valor da outra raiz da equação.

7.3.3 Criação de um vetor aleatório usando função

Fazer um programa que gere e imprima um vetor com 20 números inteiros. Estes números devem ser aleatórios e gerados por uma função **int geraInt(int limiteSuperior)**. Esta função gera aleatoriamente um número inteiro entre 0 e *limiteSuperior*.

7.3.4 Concatenação de 2 vetores ordenados em um 3º também ordenado

Fazer um programa que gere aleatoriamente dois vetores com 20 números inteiros em ordem crescente. Em seguida, é gerado um terceiro vetor com os 40 elementos dos dois primeiros, também em ordem crescente.

Exemplo:

vetorUm = {1, 4, 8, 8, 10}

vetorDois = {0, 1, 5, 9, 11}

`vetorResultado = {0, 1, 1, 4, 5, 8, 8, 9, 10, 11}`

Uma forma de gerar o vetor já ordenado é usar a função **geraInt** do exercício 7.3.3 e sempre somar o número gerado ao valor anterior. Assim, o elemento $i + 1$ será igual ao elemento $i + \text{geraInt}(\dots)$. O limite superior usado na função **geraInt()** não será respeitado nestes vetores gerados, mas não se preocupe com este fato.

A sugestão é que o programa vá percorrendo os dois vetores simultaneamente (mas não sincronizadamente) e vá transferindo valores para o **vetorResultado** de forma a manter a ordenação.

Resposta: programa 9.22.

7.3.5 Achar o maior, menor, média, soma e produto dos elementos do vetor, usando funções

Faça um programa com um menu contendo as seguintes opções: gera vetor; exhibe vetor; acha maior; acha menor; calcula média dos elementos; calcula soma dos elementos; calcula produto dos elementos; sai do programa.

Considere que o vetor em questão tem 15 elementos reais.

Implemente cada opção do menu através da chamada de uma função específica.

7.3.6 Produto de matrizes

Faça um programa que gere duas matrizes, uma 4×6 e outra 6×4 , calcule e exiba o produto delas.

Obs.: neste momento, sugiro não usar funções (exceto a função **geraInt()**).

Resposta: programa 9.23.

7.3.7 Ordenação de vetor

Faça um programa que gere um vetor de números inteiros aleatórios e faça a ordenação dos valores em ordem crescente.

Resposta: programa 9.24.

Capítulo 8

struct (Estrutura)

8.1 struct : conceitos iniciais

Em C, uma estrutura é uma coleção de variáveis agrupadas “dentro de outra variável”. Imagine que você precisa gravar vários dados de um cliente (codigo, nome, endereço, telefone, CPF, *e-mail*, limite de crédito). É interessante que você crie uma **estrutura** (**struct**) e, dentro dela, crie variáveis para cada um dos **campos** (código, nome, endereço, ...).

No exemplo dado acima, a estrutura seria a seguinte:

```
struct TipoCliente {  
    int codigo;  
    char nome[50];  
    char endereco[100];  
    char telefone[12];  
    char CPF[11];  
    char eMail[40];  
    float limiteDeCredito;  
};
```

Observe que dentro da **estrutura** temos várias variáveis de **tipos primitivos** (int, vetores de char e float).

Veja que o enunciado termina com ponto e vírgula (;).

Na verdade, no exemplo acima, ainda não foi criada a variável em si. O que foi feito foi a definição (“criação”) de um **novo tipo** chamado **TipoCliente** que contém todos aqueles campos.

Para se declarar uma variável efetivamente, é preciso um enunciado como o que mostramos a seguir:

```
struct TipoCliente cliente;
```

Aí sim, passamos a ter uma variável chamada **cliente** que é do tipo **TipoCliente**.

Você pode inclusive declarar **mais de uma variável** deste tipo:

```
struct TipoCliente cliente, clienteEspecial, clienteInativo;
```

A declaração das variáveis pode ser feita junto com a criação do tipo. No caso acima teríamos:

```
struct TipoCliente {
    int codigo;
    char nome[50];
    char endereco[100];
    char telefone[12];
    char CPF[11];
    char eMail[40];
    float limiteDeCredito;
} cliente, clienteEspecial, clienteInativo;
```

Se você precisar de apenas uma variável do tipo criado, não precisa incluir o seu nome. Veja o exemplo adaptado a este caso:

```
struct {
    int codigo;
    char nome[50];
    char endereco[100];
    char telefone[12];
    char CPF[11];
    char eMail[40];
    float limiteDeCredito;
} cliente;
```

Temos, então, como forma geral da definição de uma estrutura, esta mostrada a seguir:

```
struct NomeDoNovoTipo {
    tipo nomeDaVariável;
    tipo nomeDaVariável;
    ...
    tipo nomeDaVariável;
} variavelUm, variavelDois, ... , variavelN;
```

8.2 Acessando os elementos (campos) da estrutura

O acesso a um **campo da estrutura** se dá através do **nome da variável**, um **ponto** e o **nome do campo**.

Nos exemplos acima, teríamos:

```
cliente.codigo = 123;
```

```
clienteEspecial.limiteDeCredito = 1200.50;
```

```
gets(clienteInativo.nome);
```

De um modo geral:

```
nomeDaVariávelDoTipoEstrutura.nomeDoCampo
```

8.3 Vetores de estruturas

Depois de declarado o tipo da estrutura, basta declarar o vetor com a quantidade de elementos desejada. Ainda com relação ao exemplo da seção 8.1, veja como seria a criação de um vetor para armazenar até 100 clientes:

```
struct TipoCliente bancoDeClientes[100];
```

Para acessar um campo de um cliente em especial, combine os conceitos já estudados de vetores com os conceitos aqui apresentados:

Por exemplo, o código do trigésimo cliente é acessado através do enunciado a seguir:

```
bancoDeClientes[29].codigo
```

8.4 Exercícios Propostos

8.4.1 Cadastro de produtos

Faça um programa que permita cadastrar, consultar e modificar uma base de 15 produtos. Cada produto contém os seguintes atributos: código, nome, preço, quantidade em estoque.

Capítulo 9

Solução dos Exercícios

9.1 Respostas dos exercícios do Capítulo 1

Algoritmo/Programa 9.1: Resposta do exercício 1.6.10

```
1 //prog033.c
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <math.h>
6 int main () {
7     float x,y,r,e; //x-angulo; y-medida da parede; r-radianos e- medida da escada
8     printf ("O angulo eh:");
9     scanf ("%f", &x);
10    printf ("A medida da parede eh:");
11    scanf ("%f",&y);
12    r=x*(3.14/180);
13    e=y/sin(r);
14    printf ("A medida da escada eh: %.1f\n",e);
15 }
```

9.2 Respostas dos exercícios do Capítulo 2

Algoritmo/Programa 9.2: Resposta do exercício 2.3.1

```
1 //prog010.c
2
3 #include <stdio.h>
4
5 int main() {
```

```

6      int numeroGerado, palpite;
7
8      srand ( time(NULL) );    //faz número gerado variar a cada execução do programa
9
10     numeroGerado = rand();    //gera um número aleatório
11
12     //Tire o comentário da linha abaixo para ver o número antes de tentar
13     //printf("\nNúmero gerado: %d", numeroGerado);
14
15     numeroGerado = numeroGerado % 10;
16     //Tire o comentário da linha abaixo para ver o número reduzido antes de tentar
17     //printf("\nResto da divisão do número gerado por 10: %d", numeroGerado);
18
19     printf("\nSeu palpite:");
20     scanf("%d",&palpite);
21
22     if ( palpite == numeroGerado ) {
23         printf("\nAcertou!!!\n");
24     }
25     else {
26         printf("\nErrou...\n");
27     }
28
29 }

```

Algoritmo/Programa 9.3: Resposta do exercício 2.3.2

```

1  //prog011.c
2
3  #include <stdio.h>
4
5  int main() {
6      int numeroDigitado;
7
8      printf("\nDigite um número: ");
9      scanf("%d",&numeroDigitado);
10
11     printf("\n%d é ", numeroDigitado);
12     if ( numeroDigitado % 2 == 0 ) {
13         printf("par.\n");
14     }
15     else {
16         printf("ímpar.\n");
17     }
18
19 }

```

Algoritmo/Programa 9.4: Resposta do exercício 2.3.3

```
1 //prog012.c
2
3 #include <stdio.h>
4
5 int main() {
6     int numeroUm, numeroDois, numeroTres;
7     int maior;
8
9     printf("\nDigite 3 números separados por espaço e tecla ENTER: ");
10    scanf("%d %d %d",&numeroUm, &numeroDois, &numeroTres);
11
12
13    if ( numeroUm > numeroDois ) {
14        if ( numeroUm > numeroTres ) //só uma linha dentro do if dispensa as chaves
15            maior = numeroUm;
16        else
17            maior = numeroTres;
18    }
19    else {
20        if ( numeroDois > numeroTres )
21            maior = numeroDois;
22        else
23            maior = numeroTres;
24    }
25
26    printf("\nO maior entre %d, %d e %d é %d\n",numeroUm, numeroDois, numeroTres, maior);
27
28    //Outra solução
29    if ( numeroUm >= numeroDois && numeroUm >= numeroTres ) {
30        maior = numeroUm;
31    }
32
33    if ( numeroDois >= numeroUm && numeroDois >= numeroTres ) {
34        maior = numeroDois;
35    }
36
37    if ( numeroTres >= numeroDois && numeroTres >= numeroUm ) {
38        maior = numeroTres;
39    }
40
41    printf("\n(Por outra lógica)\nO maior entre %d, %d e %d é %d\n",numeroUm, numeroDois,
42           numeroTres, maior);
43
44 }
```

```
1 //prog013.c
2
3 #include <stdio.h>
4
5 int main() {
6     int numero;
7
8     printf("\nDigite um número: ");
9     scanf("%d",&numero);
10
11    printf("\n%d é ",numero);
12    if ( numero > 0 ) {
13        printf("positivo\n");
14    }
15    else {
16        if ( numero < 0 )
17            printf("negativo\n");
18        else
19            printf("nulo\n");
20    }
21 }
```

Algoritmo/Programa 9.6: Resposta do exercício 2.3.5

```
1 //prog014.c
2
3 #include <stdio.h>
4 #include <math.h>
5
6 //No Linux, compile com "gcc -lm" para carregar a biblioteca matemática com a função sqrt
7 //O programa requer ajustes quando a > 0 e b = c = 0.
8
9 int main() {
10     float a, b, c;
11     float delta, raizUm, raizDois;
12
13     printf("\nSeja a equação do segundo grau na forma: a x2 + b x + c = 0 \n");
14     printf("Digite os coeficientes a, b e c:");
15     printf("\na = ");
16     scanf("%f",&a);
17     if ( a == 0 ) {
18         printf("\n0 coeficiente \"a\" não pode ser nulo.\n0 programa será encerrado.\n");
19         return(0); //return encerra uma função. No caso da main, encerra o programa.
20     }
21     printf("\nb = ");
22     scanf("%f",&b);
23     printf("\nc = ");
```

```

24     scanf("%f",&c);
25
26     delta = pow(b, 2) - 4 * a * c;
27
28     if ( delta < 0 )
29         printf("\nDelta negativo, a equação \".1f x2 + (.1f) x + (.1f)\\" não tem
30             raízes reais.\n",a,b,c);
31     else {
32         if ( delta == 0 ) {
33             raizUm = raizDois = (-1 * b) / (2 * a);
34             printf("\nDelta = 0, a equação \".1f x2 + (.1f) x + (.1f)\\" tem uma
35                 raiz que vale %f.\n",a,b,c,raizUm);
36         }
37         else {
38             raizUm = ((-1 * b) + sqrt(delta)) / (2 * a);
39             raizDois = ((-1 * b) - sqrt(delta)) / (2 * a);
40             printf("\nA equação \".1f x2 + (.1f) x + (.1f)\\" tem duas raízes
41                 reais: %f e %f\n",a,b,c,raizUm,raizDois);
42         }
43     }
44 }

```

Algoritmo/Programa 9.7: Resposta do exercício 2.3.6

```

1  //prog015.c
2
3  #include <stdio.h>
4
5  int main() {
6      int numeroUm, numeroDois, soma, resultadoFinal;
7
8      printf("\nDigite dois números separados por espaço: ");
9      scanf("%d %d",&numeroUm,&numeroDois);
10
11     soma = numeroUm + numeroDois;
12
13     if ( soma > 20 )
14         resultadoFinal = soma + 8;
15     else
16         resultadoFinal = soma - 5;
17
18     printf("\nResultado final: %d\n",resultadoFinal);
19 }

```

Algoritmo/Programa 9.8: Resposta do exercício 2.3.7

```

1  //prog016.c
2
3  #include <stdio.h>

```

```
4
5 int main() {
6     float salario, valorDoEmprestimo, valorDaPrestacao;
7     int numeroDePrestacoes;
8
9     printf("\nSalário: ");
10    scanf("%f",&salario);
11
12    printf("Valor do empréstimo: ");
13    scanf("%f",&valorDoEmprestimo);
14
15    printf("Número de prestações: ");
16    scanf("%d",&numeroDePrestacoes);
17
18    if ( salario <= 0 | valorDoEmprestimo <=0 | numeroDePrestacoes <=0 ) {
19        printf("\nNenhum dos valores pode ser nulo ou negativo.\nPrograma será
20            encerrado.\n");
21        return(0);
22    }
23
24    valorDaPrestacao = valorDoEmprestimo / (float)numeroDePrestacoes;
25
26    // O uso de 0.3f abaixo especifica um float. Se usar apenas 0.3, é tratado como double
27    // e falha na comparação do >
28    if ( valorDaPrestacao > salario * 0.3f )
29        printf("\n Empréstimo não autorizado.\n");
30    else
31        printf("\n Empréstimo autorizado.\n");
32 }
```

9.3 Respostas dos exercícios do Capítulo 3

Algoritmo/Programa 9.9: Resposta do exercício 3.3.1

```
1 //prog019.c
2
3 #include <stdio.h>
4
5 int main() {
6     int planetaEscolhido;
7     float pesoNaTerra, gravidadeRelativa, pesoNoPlaneta;
8
9     printf("\nPlanetas");
10    printf("\n1 - Mercúrio");
11    printf("\n2 - Vênus");
12    printf("\n3 - Marte");
```

```
13     printf("\n4 - Júpiter");
14     printf("\n5 - Saturno");
15     printf("\n6 - Urano");
16     printf("\n7 - Netuno\n");
17     printf("\nDigite o número do planeta escolhido: ");
18     scanf("%d",&planetaEscolhido);
19
20     printf("\n\nInforme seu peso na Terra: ");
21     scanf("%f",&pesoNaTerra);
22
23     switch (planetaEscolhido) {
24         case 1:
25             gravidadeRelativa = 0.376;
26             break;
27         case 2:
28             gravidadeRelativa = 0.903;
29             break;
30         case 3:
31             gravidadeRelativa = 0.380;
32             break;
33         case 4:
34             gravidadeRelativa = 2.340;
35             break;
36         case 5:
37             gravidadeRelativa = 1.160;
38             break;
39         case 6:
40             gravidadeRelativa = 1.150;
41             break;
42         case 7:
43             gravidadeRelativa = 1.190;
44             break;
45         default:
46             gravidadeRelativa = 0;
47     }
48
49     if ( gravidadeRelativa != 0 ) {
50         pesoNoPlaneta = pesoNaTerra * gravidadeRelativa;
51         printf("\n\nSeu peso no planeta escolhido é %.3f\n\n",pesoNoPlaneta);
52     }
53     else {
54         printf("\n\nErro na escolha do planeta.");
55     }
56
57 }
```

```
1 //prog020.c
2
3 #include <stdio.h>
4
5 int main() {
6     char opcao;
7
8     printf("\nQual a raiz quadrada de 49?\n");
9     printf("\n\ta) 4\n\tb) 7\n\tc) 9\n\td) 6\n\te) Nenhuma das respostas anteriores\n");
10
11     printf("\n\tSua opção: ");
12
13     //opcao = getchar();          esta linha tem o mesmo funcionamento da linha abaixo
14     scanf("%c",&opcao);
15
16     switch (opcao) {
17         case 'a':
18             printf("\n\tVocê marcou letra \"a\" e errou.\n\n");
19             break;
20         case 'b':
21             printf("\n\tVocê marcou letra \"b\" e acertou.\n\n");
22             break;
23         case 'c':
24             printf("\n\tVocê marcou letra \"c\" e errou.\n\n");
25             break;
26         case 'd':
27             printf("\n\tVocê marcou letra \"d\" e errou.\n\n");
28             break;
29         case 'e':
30             printf("\n\tVocê marcou letra \"e\" e errou.\n\n");
31             break;
32         default:
33             printf("\n\tOpção inválida.\n\n");
34     }
35 }
```

Algoritmo/Programa 9.11: Resposta do exercício 3.3.3

```
1 //prog017.c
2
3 #include <stdio.h>
4
5 int main() {
6     float operandoUm, operandoDois;
7     char operador;
8
9     printf("\nDigite o primeiro número: ");
10    scanf("%f",&operandoUm);
```

```
11
12     printf("\nDigite o segundo número: ");
13     scanf("%f",&operandoDois);
14
15     printf("\nDigite a operação (+ - * /): ");
16
17     getchar();
18     operador = getchar();
19
20     switch (operador) {
21         case '+':
22             printf("\nResultado = %.2f\n",operandoUm+operandoDois);
23             break;
24         case '-':
25             printf("\nResultado = %.2f\n",operandoUm-operandoDois);
26             break;
27         case '*':
28             printf("\nResultado = %.2f\n",operandoUm*operandoDois);
29             break;
30         case '/':
31             if ( operandoDois != 0 ) {
32                 printf("\nResultado = %.2f\n",operandoUm/operandoDois);
33             }
34             else {
35                 printf("\nDivisão por zero não efetuada.\n");
36             }
37             break;
38         default:
39             printf("\nOperação desconhecida.\n");
40     }
41 }
```

9.4 Respostas dos exercícios do Capítulo 4

Algoritmo/Programa 9.12: Resposta do exercício 4.3.1

```
1 //prog024.c
2
3 #include <stdio.h>
4
5 int main() {
6     int numeroGerado, palpite, numeroDeTentativas = 1, i;
7
8     srand ( time(NULL) );
9
10    numeroGerado = rand();
```

```
11
12     numeroGerado = numeroGerado % 10;
13
14     printf("Quantas tentativas quer fazer? ");
15     scanf("%d", &numeroDeTentativas);
16
17     for ( i = 1 ; i <= numeroDeTentativas ; i++ ) {
18
19         printf("\nTentativa número %d.\n\tSeu palpite:",i);
20         scanf("%d",&palpite);
21
22         if ( palpite == numeroGerado ) {
23             printf("\nAcertou!!!\n");
24             break; //Interrompe a execução do for
25         }
26         else {
27             printf("\nErrou...\n");
28         }
29     }
30 }
```

Algoritmo/Programa 9.13: Resposta do exercício 4.3.2

```
1 //prog025.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     int opcao, base, potencia;
8
9     printf("\n\n      *** Menu ***");
10    printf("\n  2 - Potências de 2.");
11    printf("\n  3 - Potências de 3.");
12    printf("\n  5 - Potências de 5.");
13    printf("\n 10 - Potências de 10.");
14    printf("\n 15 - Potências de 15.");
15    printf("\n  Sua opção: ");
16    scanf("%d", &opcao);
17
18    switch(opcao) {
19        case 2:
20            base = 2;
21            break;
22        case 3:
23            base = 3;
24            break;
25        case 5:
```

```

26             base = 5;
27             break;
28         case 10:
29             base = 10;
30             break;
31         case 15:
32             base = 15;
33             break;
34         default:
35             base = 10000;
36     }
37
38     if ( base != 10000 ) {
39         printf("\n Potências de %d menores que 10000: ",base);
40         for ( potencia = base ; potencia < 10000 ; potencia *= base )
41             printf("%d ",potencia);    //um só comando dispensa as chaves do for
42         printf("\n\n");
43     }
44
45 }

```

Algoritmo/Programa 9.14: Resposta do exercício 4.3.3

```

1  //prog027.c
2
3  #include <stdio.h>
4
5  int main() {
6
7      int i, a1, r, n, ai, somaDosTermos = 0, somaDosTermosPelaFormula;
8
9      printf("\n a1 = ");
10     scanf("%d",&a1);
11
12     printf("\n r = ");
13     scanf("%d",&r);
14
15     printf("\n Número de termos. n = ");
16     scanf("%d",&n);
17
18     ai = a1;
19
20     for ( i = 1 ; i <= n ; i++ ) {
21         if ( (i-1) % 10 == 0)
22             printf("\n");
23         printf("\t%d",ai);
24         somaDosTermos += ai;
25         ai += r;

```

```

26     }
27
28     ai == r;    //sai do for já com o próximo termo da PA
29
30     printf("\n Soma dos termos acumulada = %d",somaDosTermos);
31
32     somaDosTermosPelaFormula = n * (a1 + ai) / 2;
33
34     printf("\n Soma dos termos calculada = %d\n\n",somaDosTermosPelaFormula);
35
36 }

```

Algoritmo/Programa 9.15: Resposta do exercício 4.3.4

```

1  //prog028.c
2
3  #include <stdio.h>
4
5  int main() {
6
7      int i, n, termo_i_menos_2 = 0, termo_i_menos_1 = 1, termo;
8
9      printf("\n n = ");
10     scanf("%d", &n);
11
12     if ( n > 1 ) {
13         printf("\n Sequência dos %d primeiros números da sequência de Fibonacci.\n",n)
14         ;
15         printf(" %d %d",termo_i_menos_2,termo_i_menos_1);
16         for ( i = 3 ; i <= n ; i++ ) {
17             termo = termo_i_menos_1 + termo_i_menos_2;
18             printf(" %d ",termo);
19             termo_i_menos_2 = termo_i_menos_1;
20             termo_i_menos_1 = termo;
21             if ( i%15 == 0 )
22                 printf("\n");
23         }
24         printf("\n\n");
25     }
26 }

```

Algoritmo/Programa 9.16: Resposta do exercício 4.3.5

```

1  //prog029.c
2
3  #include <stdio.h>
4
5  int main() {

```

```

6     int i, j, numero;
7     printf("\n número ( >= 0 ) = ");
8     scanf("%d", &numero);
9     if ( numero >= 0 ) {
10        printf("\n Pares de números cuja soma é igual a %d.\n",numero);
11        for ( i = 0, j = numero ; i <= numero ; i++, j-- ) {
12            printf("\t %d + %d = %d \n",i, j, numero);
13        }
14        printf("\n\n");
15    }
16 }

```

Algoritmo/Programa 9.17: Resposta do exercício 4.3.6

```

1  //prog030.c
2
3  #include <stdio.h>
4
5  int main() {
6
7      int numero, soma = 0, i;
8
9      for ( i = 1 ; i <= 10 ; i++ ) {
10
11         printf(" (rodada %d, faltam %d).  Digite um número: ",i, 10-i);
12         scanf("%d", &numero);
13
14         if ( numero * numero < 225 ) {
15             soma += numero;
16         }
17     }
18
19     printf("\nSoma dos números cujos quadrados são menores que 225: %d \n\n", soma);
20
21
22 }

```

Algoritmo/Programa 9.18: Resposta do exercício 4.3.7

```

1  //prog031.c
2
3  #include <stdio.h>
4
5  int main() {
6
7      int numero, i;
8
9      printf("\n Digite um número: ");
10     scanf("%d", &numero);

```

```

11
12     printf("\n Divisores de %d: ", numero);
13
14     for ( i = 1 ; i <= numero / 2 ; i++ ) {
15
16         if ( numero % i == 0 ) {
17             printf("%d ", i);
18         }
19     }
20
21     printf("%d ", numero);
22
23     printf("\n\n");
24
25 }

```

Algoritmo/Programa 9.19: Resposta do exercício 4.3.8

```

1  //prog032.c
2
3  #include <stdio.h>
4
5  int main() {
6
7      int i, j, k;
8
9      printf("\n Impressão de (i,j)\n");
10
11     printf("Usando a solução 1:\n\n");
12     for ( i = 0 ; i < 10 ; i++ ) {
13         printf("\n");
14         for ( j = 0 ; j < 10 ; j++ ) {
15             if ( j >= i ) {
16                 printf("\t(%d, %d)", i, j);
17             }
18             else {
19                 /* imprime espaços em branco,
20                  quando o número não é, impresso,
21                  para manter a tela formatada */
22                 printf("\t    ");
23             }
24         }
25     }
26
27     printf("\n\nUsando a solução 2:\n\n");
28     for ( i = 0 ; i < 10 ; i++ ) {
29         printf("\n");
30         //O for abaixo é para manter a tela formatada

```

```
31         for ( k = 0; k < i ; k++ )
32             printf("\t");
33         for ( j = i ; j < 10 ; j++ ) {
34             printf("\t(%d, %d)", i, j);
35         }
36     }
37
38     printf("\n\n");
39
40 }
```

9.5 Respostas dos exercícios do Capítulo 5

Algoritmo/Programa 9.20: Resposta do exercício 5.3.1

```
1  //prog034.c
2
3  #include <stdio.h>
4
5  int main() {
6
7      int numeroGerado, palpito = 0;
8
9      srand ( time(NULL) );
10
11     numeroGerado = rand();
12
13     numeroGerado = numeroGerado % 100;
14
15     while ( palpito != -1 ) {
16
17         printf("\n Digite seu palpito: ");
18         scanf("%d",&palpito);
19
20         if ( palpito == numeroGerado ) {
21             printf("\nAcertou!!!\n");
22             break; //Interrompe a execução do while
23         }
24         else {
25             if ( palpito != -1 )
26                 printf("\nErrou...\n");
27         }
28     }
29 }
```

9.6 Respostas dos exercícios do Capítulo 6

Algoritmo/Programa 9.21: Resposta do exercício 6.4.1

```
1 //prog038.c
2
3 #include <stdio.h>
4
5 int main() {
6
7     int i;
8
9     float somaDasNotas = 0, mediaDaTurma;
10
11    float notas[8];
12
13    //Lê notas
14    printf("\nDigite as notas da turma:\n");
15    for ( i = 0 ; i < 8 ; i ++ ) {
16        scanf("%f",&notas[i]);
17    }
18
19    //Soma notas
20    for ( i = 0 ; i < 8 ; i ++ ) {
21        somaDasNotas += notas[i];
22    }
23
24    //Calcula e exibe a média
25    mediaDaTurma = somaDasNotas / 8;
26    printf("\nMédia da turma = %.2f\n",mediaDaTurma);
27
28 }
```

9.7 Respostas dos exercícios do Capítulo 7

Algoritmo/Programa 9.22: Resposta do exercício 7.3.4

```
1 //prog050.c
2
3 #include <stdio.h>
4
5 int geraInt(int limiteSuperior) {
6     int numeroGerado = rand();
7     numeroGerado %= limiteSuperior;
8     return(numeroGerado);
9 }
```

```
10
11 int main() {
12
13     int vet1[20], vet2[20], vet3[40], i, j, k;
14
15     srand(time(NULL));
16
17     vet1[0] = geraInt(100);
18     for ( i = 1 ; i < 20 ; i++ ) {
19         vet1[i] = vet1[i-1] + geraInt(100);
20     }
21
22     vet2[0] = geraInt(100);
23     for ( i = 1 ; i < 20 ; i++ ) {
24         vet2[i] = vet2[i-1] + geraInt(100);
25     }
26
27     printf("\n\n Vetor 1      ");
28     for ( i = 0 ; i < 20 ; i++ ) {
29         printf("%d ",vet1[i]);
30     }
31
32     printf("\n\n Vetor 2      ");
33     for ( i = 0 ; i < 20 ; i++ ) {
34         printf("%d ",vet2[i]);
35     }
36
37     i=0;
38     j=0;
39     for ( k = 0 ; k < 40 ; k++ ) {
40         if (i==20) {
41             vet3[k] = vet2[j];
42             j++;
43             continue; // Volta ao topo do for
44         }
45         if (j==20) {
46             vet3[k] = vet1[i];
47             i++;
48             continue;
49         }
50
51
52         if (vet1[i] < vet2[j]) {
53             vet3[k] = vet1[i];
54             i++;
55         }
56         else {
57             vet3[k] = vet2[j];
```

```

58             j++;
59         }
60     }
61
62     printf("\n\n Vetor 3      ");
63     for ( i = 0 ; i < 40 ; i++ ) {
64         printf("%d ",vet3[i]);
65     }
66     printf("\n\n");
67 }

```

Algoritmo/Programa 9.23: Resposta do exercício 7.3.6

```

1  //prog048.c
2
3  #include <stdio.h>
4
5  int geraInt(int);
6  void imprimeMatriz(int, int, int);
7  int multiplicaMatrizes(int, int);
8
9  int main(){
10
11     srand(time(NULL));
12
13     int matrizUm[4][6], matrizDois[6][4], matrizProduto[4][4];
14
15     int i, j, k;
16
17     // Geração da matrizUm
18     for ( i = 0 ; i < 4 ; i++ ) {
19         for ( j = 0 ; j < 6 ; j++ ) {
20             matrizUm[i][j] = geraInt(10);
21         }
22     }
23
24     printf("\n\nMatriz Um\n");
25     for ( i = 0 ; i < 4 ; i++ ) {
26         for ( j = 0 ; j < 6 ; j++ ) {
27             printf("%d\t",matrizUm[i][j]);
28         }
29         printf("\n");
30     }
31
32     // Geração da matrizDois
33     for ( i = 0 ; i < 6 ; i++ ) {
34         for ( j = 0 ; j < 4 ; j++ ) {
35             matrizDois[i][j] = geraInt(10);

```

```

36         }
37     }
38
39     printf("\n\nMatriz Dois\n");
40     for ( i = 0 ; i < 6 ; i++ ) {
41         for ( j = 0 ; j < 4 ; j++ ) {
42             printf("%d\t",matrizDois[i][j]);
43         }
44         printf("\n");
45     }
46
47     //Cálculo da matrizProduto
48     for ( i = 0; i < 4; i++ ) {
49         for ( j = 0; j < 4; j++ ) {
50             matrizProduto[i][j] = 0;
51             for ( k = 0; k < 6; k++ ) {
52                 matrizProduto[i][j] += matrizUm[i][k] * matrizDois[k][j];
53             }
54         }
55     }
56
57     printf("\n\nMatriz Produto\n");
58     for ( i = 0 ; i < 4 ; i++ ) {
59         for ( j = 0 ; j < 4 ; j++ ) {
60             printf("%d\t",matrizProduto[i][j]);
61         }
62         printf("\n");
63     }
64
65 }
66
67 int geraInt(int limiteSuperior)
68 {
69     int intGerado;
70     intGerado = rand();
71     intGerado = intGerado % limiteSuperior;
72     return (intGerado);
73 }

```

Algoritmo/Programa 9.24: Resposta do exercício 7.3.7

```

1 //prog049.c
2
3 #include <stdio.h>
4
5 int quantidade(){
6     int quantidadeDeValores;
7     puts("Entre com a quantidade de numeros que deseja ordenar: ");

```

```
8     scanf("%d",&quantidadeDeValores);
9     return(quantidadeDeValores);
10 }
11
12 int geraInt(int limiteSuperior) {
13     int numeroGerado = rand();
14     numeroGerado %= limiteSuperior;
15     return(numeroGerado);
16 }
17
18 void imprime(int argNUmeros[], int argQ) {
19     int i;
20     for ( i = 0 ; i < argQ ; i++ )
21         printf("%d ",argNUmeros[i]);
22 }
23
24 int main(){
25     int i, j, q, temp;
26     q = quantidade(); //quantidade de numeros do vetor
27
28     if ( q <= 0 ) {
29         printf("\nERRO: quantidade negativa ou nula.\n");
30         return(0);
31     }
32
33     int numeros[q];
34
35     srand(time(NULL));
36
37     //Gera vetor
38     for ( i = 0 ; i < q ; i++)
39         numeros[i] = geraInt(100);
40
41     printf("\nVetor gerado:\n");
42     imprime(numeros, q);
43     printf("\n");
44
45     //Ordena vetor em ordem crescente
46     for ( j = q-1 ; j > 0; j-- ) {
47         for ( i = 0 ; i < j ; i++) {
48             if (numeros[i] > numeros[i+1]) {
49                 temp = numeros[i];
50                 numeros[i] = numeros[i+1];
51                 numeros[i+1] = temp;
52             }
53         }
54     }
55 }
```

```
56     printf("\nVetor ordenado:\n");
57     imprime(numeros, q);
58     printf("\n");
59
60 }
```

Referências Bibliográficas

- [1] Anita Lopes and Guto Garcia. *Introdução à programação - 500 algoritmos resolvidos*. Elsevier, Rio de Janeiro, 1. edition, 2002.
- [2] Herbert Schildt. *Turbo C: guia do usuário*. McGraw-Hill, São Paulo, 2. edition, 1988.
- [3] Herbert Schildt. *C, completo e total*. Pearson Makron Books, São Paulo, 3. edition, 1997.