PROGRAMAÇÃO A

Estrutura de Repetição

INTRODUÇÃO

- Trechos de algoritmos e consequentemente comandos de um determinado programa que precisam ser repetidos para realizar algum tipo de leitura de dados ou cálculo aritmético são chamados de **laços de repetição.**
- A **estrutura de repetição** é importante para resolver problemas computacionais que exigem a leitura de muitos dados, a soma e contagem de elementos, manipulação de elementos de um **vetor** e **matriz,** entre outras tarefas.
- Os **laços de repetição** são conhecidos também por sua tradução em inglês: ***loops*** ou ***looping***. Ganham esse nome por lembrarem uma execução finita em círculos, que depois segue seu curso normal.

ACUMULADORES E CONTADORES

Um **acumulador** é uma variável que ocorre em ambos os lados de uma atribuição e que, antes de ser usada pela primeira vez, é iniciada com um valor específico. Por exemplo, efetuando a atribuição **a = 3**, iniciamos a variável **a** com o valor **3**; então, quando a atribuição **a = a + 2** é executada, essa variável passa a ter o valor **5**. Note que, se o valor inicial de **a** não é definido pela primeira atribuição, não é possível determinar o seu valor após a execução da segunda atribuição.

Pelo fato de operações com acumuladores serem tão comuns em programação, a linguagem C oferece um conjunto de **operadores aritméticos de atribuição** que permitem escrever expressões com acumuladores de uma forma mais compacta. Por exemplo, usando o operador aritmético de atribuição **+=**, a operação de atribuição **a = a + 2** pode ser codificada como **a += 2** e a operação de atribuição **a = a + 2 * b** pode ser codificada como **a += 2 * b**.

Operação	Fluxograma	Programa em C
soma	a ← a + expr	a += expr
subtração	a ← a - expr	a -= expr
multiplicação	a ← a * expr	a *= expr
divisão real	a ← a / expr	a /= expr
divisão inteira	a ← a div expr	a /= expr
resto da divisão	a ← a mod expr	a %= expr

Tabela 1 - Operadores aritméticos de atribuição em C.

ACUMULADORES E CONTADORES

Um **contador** é um tipo de acumulador cujo valor aumenta, ou diminui, de 1 em 1.

Operadores de **incremento** e **decremento** podem ser usados na forma **prefixa** (++c e --c) ou **posfixa** (c++ e c--). Na forma **prefixa**, o valor da variável é modificado e depois usado; na forma **posfixa**, o valor da variável é usado e depois modificado. Assim, por exemplo, se c vale 3, a operação d = ++c armazena 4 em d; enquanto a operação d = c++ armazena 3 em d. Em ambos os casos, o valor final de c é 4.

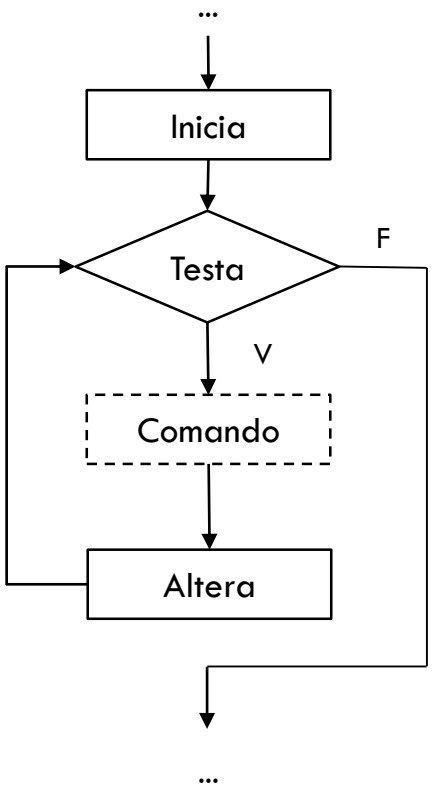
Operação	Fluxograma	Programa em C
incremento	c ← c + 1	c++
decremento	c ← c - 1	c--
Forma prefixa	++c e --c	
Forma posfixa	c++ e c--	

Tabela 2 - Operadores de incremento e decremento em C.

ESTRUTURA DE REPETIÇÃO CONTADA

A **estrutura de repetição contada** serve para repetir a execução de um comando por um determinado número de vezes. Para saber quando o total de repetições desejadas já foi atingido, a estrutura de repetição usa um **contador**. Nessa estrutura:

- Primeiramente o contador é **iniciado** com um valor específico.
- Depois o contador é **testado**: se o total de repetições ainda não foi atingido, a repetição continua; caso contrário, ela termina.
- A cada nova repetição o contador é **alterado**.



ESTRUTURA DE REPETIÇÃO CONTADA

A **estrutura de repetição contada** em linguagem C é codificada da seguinte forma:

```
for (inicia; testa; altera)
      comando a ser repetido;
```

Note que para executar mais de um comando, é preciso usar um par de chaves.

```
for (inicia; testa; altera)
{
      comando-1;
       ...
      comando-n;
}
```

ESTRUTURA DE REPETIÇÃO CONTADA

Problema 1: Dado um número inteiro, exiba a sua tabuada.

Este problema mostra um caso em que a estrutura de repetição contada é útil para produzir uma saída em vídeo, composta de várias linhas. A Figura 1 mostra um exemplo de tela de execução para este problema. A Figura 2 apresenta o algoritmo representado por um fluxograma para resolver este problema.

```
Número? 5 ↵
Tabuada do 5
5 x  1 =  5
5 x  2 = 10
5 x  3 = 15
5 x  4 = 20
5 x  5 = 25
5 x  6 = 30
5 x  7 = 35
5 x  8 = 40
5 x  9 = 45
5 x 10 = 50
```

Figura 1 - Tela de execução

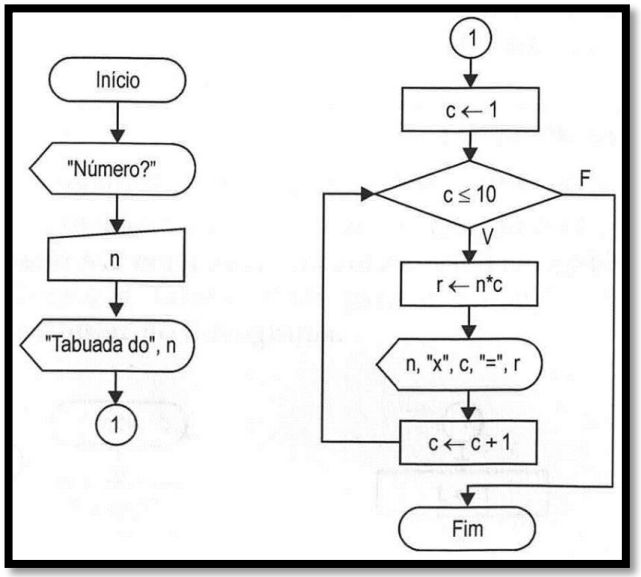


Figura 2 - Fluxograma

ESTRUTURA DE REPETIÇÃO CONTADA

Problema 1: Dado um número inteiro, exiba a sua tabuada.

O código correspondente em linguagem C para resolver este problema é mostrado na Figura 3.

```c
/* Programa da Tabuada com o laço de repetição FOR. */

#include <stdio.h>

int main()
{
    int numero, contador, resultado;

    printf("Digite um numero para obter a sua tabuada: ");
    scanf("%d%*c", &numero);

    printf("\n****** TABUADA DO NUMERO %d ******\n\n", numero);

    for(contador=1; contador<=10; contador++)
    {
        resultado = numero * contador;
        printf("%d * %2d = %3d\n", numero, contador, resultado);
    }

    return 0;
}
```

Figura 3 - Programa em C

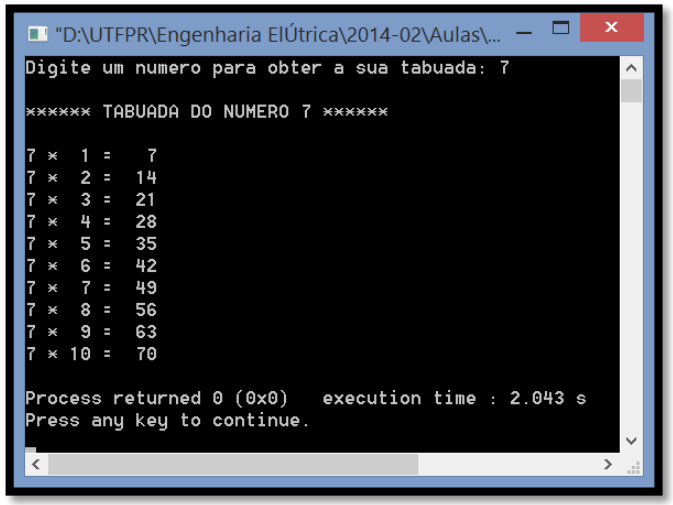


Figura 4 - Programa em execução

tabuada_com_for.c

ESTRUTURA DE REPETIÇÃO CONTADA (DECRESCENTE)

A **estrutura de repetição contada** também pode ser usada com um contador que, em vez de aumentar de 1 em 1, **diminui de 1 em 1**.

Problema 2: Dado um número *n*, exiba uma contagem regressiva de *n* até *0*.

Este problema mostra um caso em que a estrutura de repetição contada decrescente é útil. A Figura 5 mostra um exemplo de tela de execução para este problema. A Figura 6 apresenta o algoritmo representado por um fluxograma para resolver este problema.

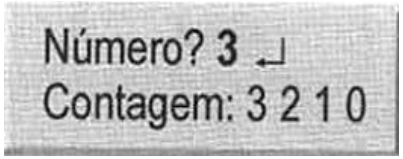


Figura 5 - Tela de execução

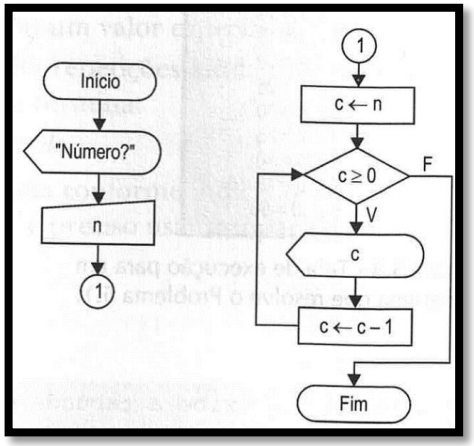


Figura 6 - Fluxograma

ESTRUTURA DE REPETIÇÃO CONTADA (DECRESCENTE)

Problema 2: Dado um número *n*, exiba uma contagem regressiva de *n* até *0*.

O código correspondente em linguagem C para resolver este problema é mostrado na Figura 7.

```c
/* Programa da Contagem Regressiva com o laço de repetição FOR. */

#include <stdio.h>

int main()
{
    int numero, contador;

    printf("Digite um numero: ");
    scanf("%d%*c", &numero);

    printf("\nContagem regressiva: ");

    for(contador=numero; contador>=0; contador--)
        printf("%d ", contador);

    printf("*** KABOOOOOOMMM ***\n");

    return 0;
}
```

Figura 7 - Programa em C

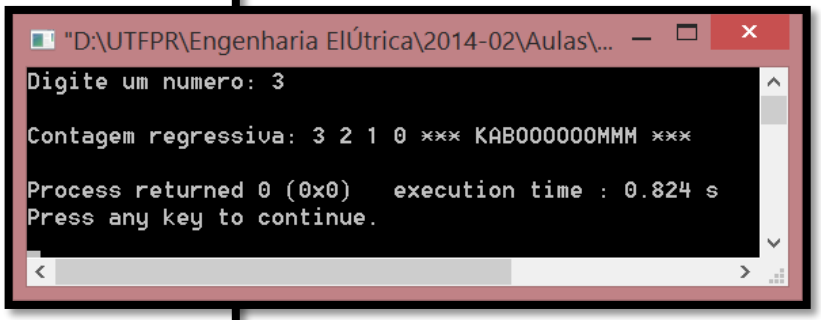


Figura 8 - Programa em execução

contagem_regressiva.c

ESTRUTURAS DE REPETIÇÃO ENCAIXADAS

Estruturas de repetição contadas podem ser encaixadas para formar padrões de repetição mais complexos.

Problema 3: Dado um número natural *n*, desenhe um quadrado *n* x *n*.

A Figura 10 apresenta o fluxograma para resolver este problema.

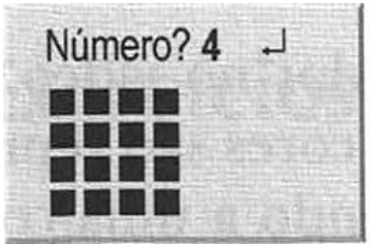


Figura 9 - Tela de execução

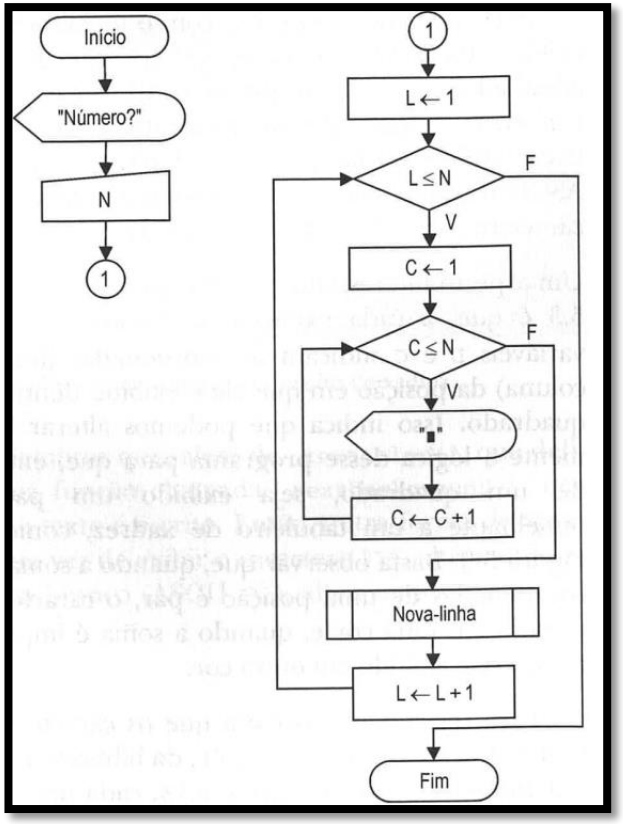


Figura 10 - Fluxograma

ESTRUTURAS DE REPETIÇÃO ENCAIXADAS

Problema 3: Dado um número natural *n*, desenhe um quadrado *n* x *n*.

O código correspondente em linguagem C para resolver este problema é mostrado na Figura 11.

```c
/* Programa que desenha um quadrado n x n usando o FOR. */

#include <stdio.h>

int main()
{
    int numero, linha, coluna;

    printf("Digite um numero: ");
    scanf("%d%*c", &numero);

    printf("\n");

    for(linha=1; linha<=numero; linha++)
    {
        for(coluna=1; coluna<=numero; coluna++)
        {
            printf("%c ", 223);
        }

        printf("\n");
    }

    return 0;
}
```

Figura 11 - Programa em C

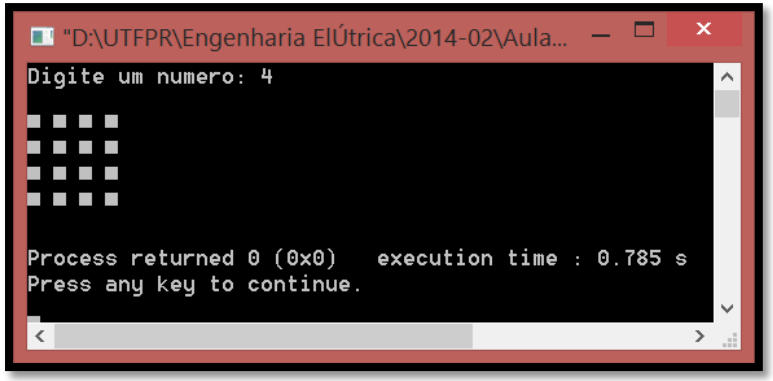


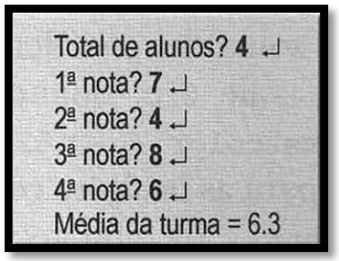
Figura 12 - Programa em execução

quadrado.c

EXERCÍCIOS

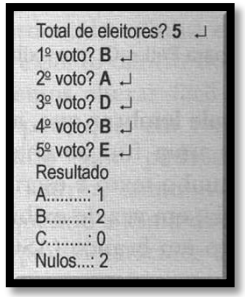
Para cada problema a seguir, codifique um programa correspondente em linguagem C e teste o programa usando a ferramenta *Code::Blocks*.

1. Dadas as notas dos alunos de uma turma, informe a média da turma. O programa deve funcionar como indicado na imagem do Problema 1.
2. Dadas as idades dos pacientes de uma clínica, informe a idade daquele mais idoso. Considere que todas as idades são distintas e que número de pacientes é informado pelo usuário, no momento da execução do programa.
3. Dados um capital, uma taxa de juros mensal e um período em meses, informe o montante ao final de cada mês. O programa deve funcionar como mostra a imagem do Problema 3.
4. Em uma eleição há três candidatos, identificados como A, B e C. Dados os votos dos eleitores, informe o resultado da eleição, conforme exemplificado na imagem do Problema 4.



Problema 1

Problema 3



Problema 4

exercicio_for_1.c

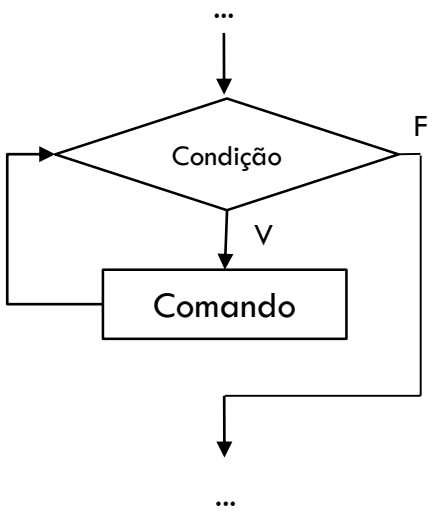
exercicio_for_2.c

exercicio_for_3.c

exercicio_for_4.c

ESTRUTURA DE REPETIÇÃO COM PRECONDIÇÃO

A **estrutura de repetição com precondição** serve para executar um ou vários comandos, repetidamente, enquanto uma determinada condição for verdadeira. Observe que, como a **condição é avaliada antes de o comando ser executado, se ela for inicialmente falsa, o comando dentro da repetição jamais é executado.**




```
while (condição)
    comando;
```

```
while (condição)
{
    comando-1;
    ...
    comando-n;
}
```

Padrão de codificação da estrutura de repetição com precondição em linguagem C.

* Note que para executar mais de um comando, é preciso usar um par de chaves.

ESTRUTURA DE REPETIÇÃO COM PRECONDIÇÃO

Problema 4: Dado um número inteiro qualquer, exiba seus dígitos.

Este problema mostra um caso em que a estrutura de repetição com precondição é útil. A Figura 13 mostra um exemplo de tela de execução para este problema. A Figura 14 apresenta o algoritmo representado por um fluxograma para resolver este problema.

```
Número? 5713 ↵
Dígitos:
3
1
7
5
```

Figura 13 - Tela de execução

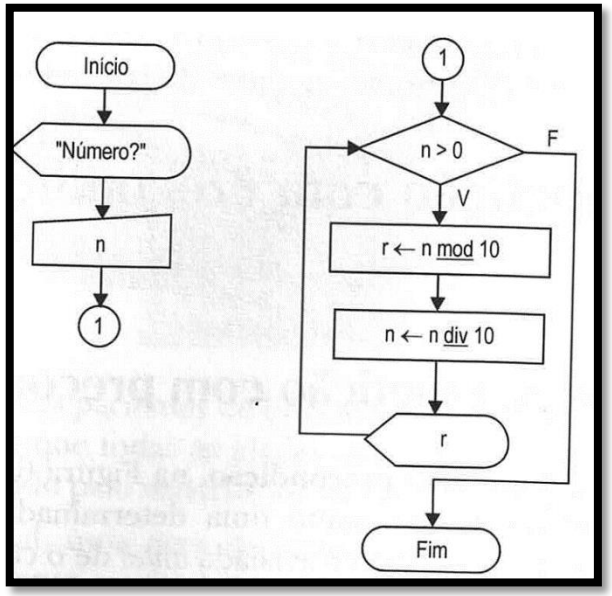


Figura 14 - Fluxograma

ESTRUTURA DE REPETIÇÃO COM PRECONDIÇÃO

Problema 4: Dado um número inteiro qualquer, exiba seus dígitos.

O código correspondente em linguagem C para resolver este problema é mostrado na Figura 15.

```c
/* Problema: Dado um número inteiro qualquer, exiba seus dígitos. */

#include <stdio.h>

int main()
{
    int numero, digito;

    printf("Digite um numero: ");
    scanf("%d%*c", &numero);

    printf("\nDigitos:\n");

    while(numero > 0)
    {
        digito = numero % 10;   // resto da divisão por 10
        numero = numero / 10;   // divide por 10 para obter o p
        printf("%d\n", digito);
    }

    return 0;
}
```

Figura 15 - Programa em C

digitos.c

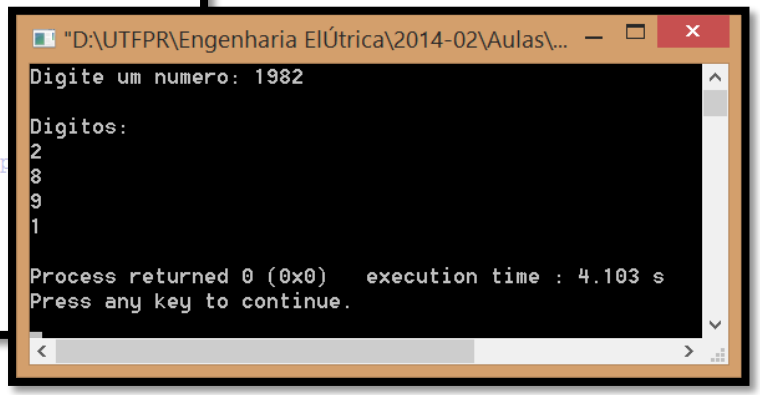


Figura 16 - Programa em execução

ESTRUTURA DE REPETIÇÃO COM PRECONDIÇÃO

Problema 5: Em um banco, as contas são identificadas por um número de conta com dígito verificador. Esse dígito verificador é calculado do seguinte modo: primeiramente somando todos os dígitos do número de conta, depois dividimos a soma por 10 e tomamos o resto. Por exemplo, se o número da conta for 5713, temos a soma 3+1+7+5 = 16; dividindo 16 por 10, temos o resto 6, que é seu dígito verificador. Dado um número de conta, informe o dígito verificador correspondente.

```
Número? 5713 ↵
DV = 6
```

Figura 17 - Tela de execução

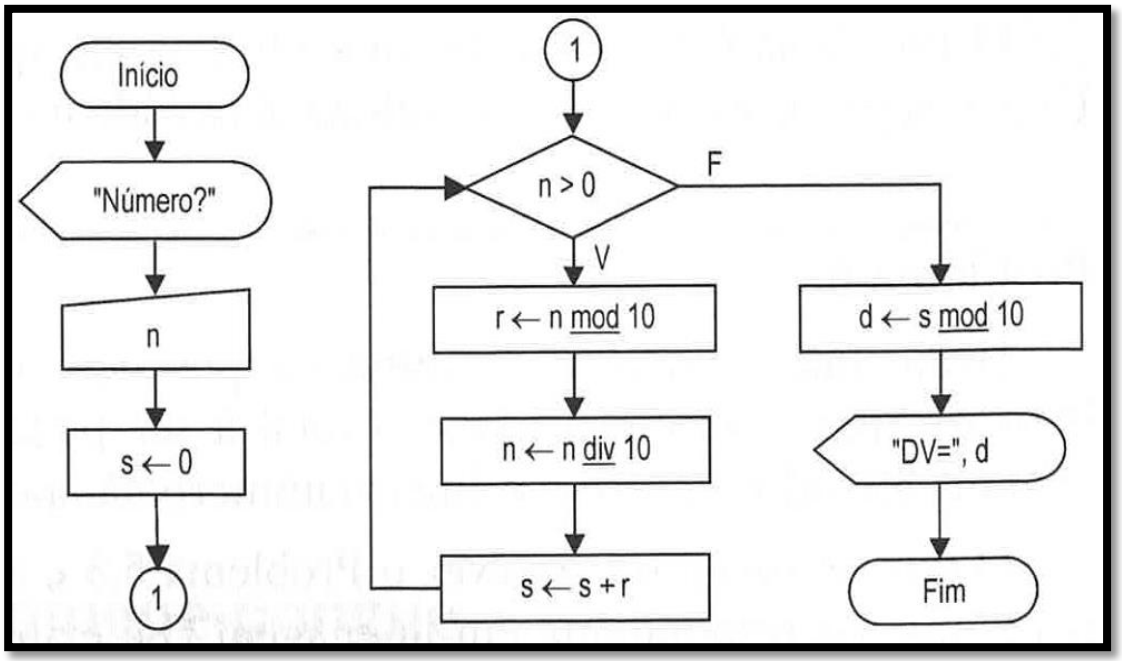


Figura 18 - Fluxograma

ESTRUTURA DE REPETIÇÃO COM PRECONDIÇÃO

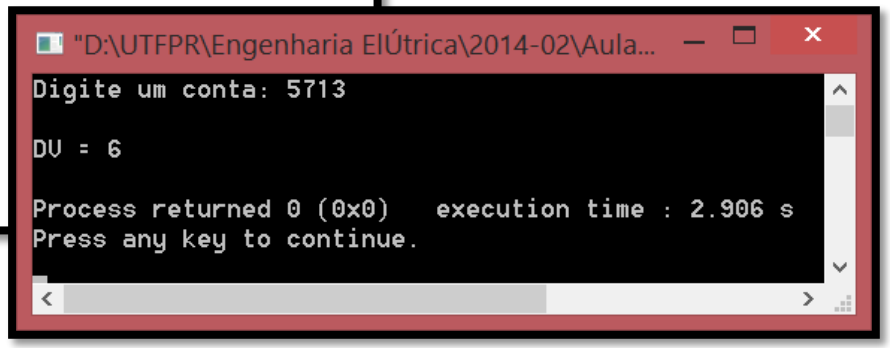
Uma possível solução para o problema proposto

```c
#include <stdio.h>

int main()
{
    int numero, digito, dv, soma=0;

    printf("Digite um conta: ");
    scanf("%d%*c", &numero);

    while(numero > 0)
    {
        digito = numero % 10;    // resto da divisão por 10
        numero /= 10;            // equivale a numero = numero / 10
        soma += digito;          // equivale a soma = soma + digito
    }

    dv = soma % 10;
    printf("\nDV = %d\n", dv);

    return 0;
}
```

Figura 19 - Programa em C

digito_verificador.c

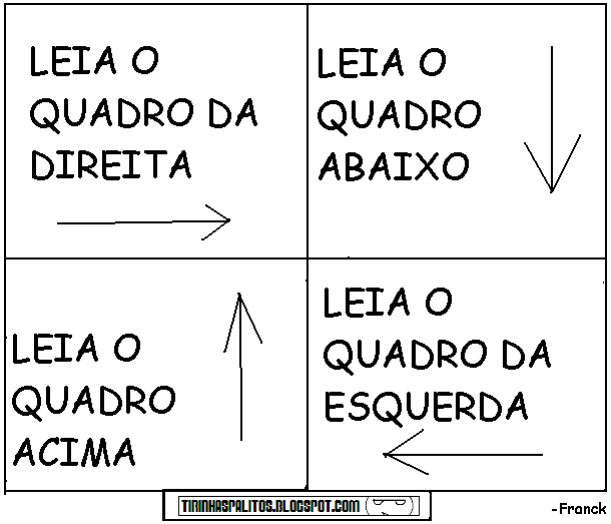
Figura 20 - Programa em execução



REPETIÇÃO COM TERMINAÇÃO FORÇADA

Em linguagem C, quando usamos o comando **while (1) { ... }**, criamos uma repetição cuja condição de repetição é sempre verdadeira, ou seja, uma **repetição infinita**; porém com o comando **break**, podemos forçar o término de qualquer repetição.

Repetição com terminação forçada não é um padrão convencional de programação estruturada e, sendo assim, devemos evitar o seu uso quando construímos fluxogramas a serem implementados em outras linguagens.



REPETIÇÃO COM TERMINAÇÃO FORÇADA

Problema 6: Dada uma sequencia de números positivos, representando os valores dos itens de uma compra, informe o total a ser pago. Considere que a sequencia termina quando aparece o primeiro número não positivo.

Figura 21 - Tela de execução

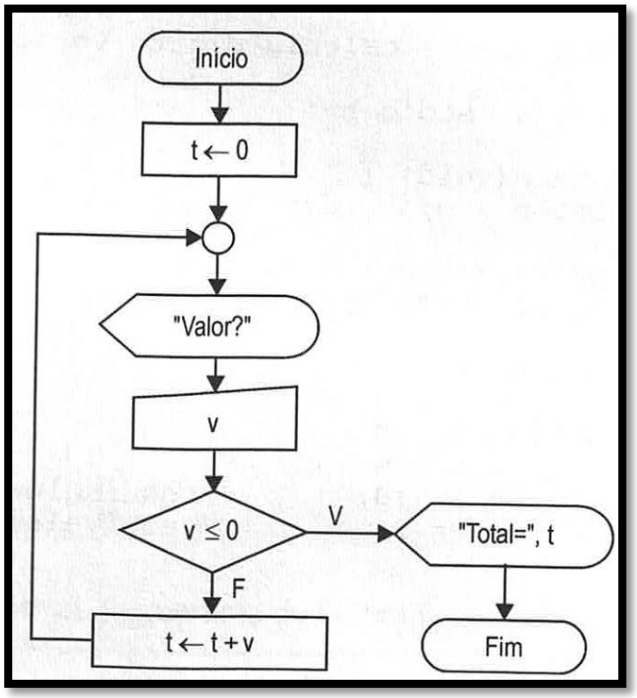


Figura 22 - Fluxograma

REPETIÇÃO COM TERMINAÇÃO FORÇADA

Problema 6: Dada uma sequencia de números positivos, representando os valores dos itens de uma compra, informe o total a ser pago. Considere que a sequencia termina quando aparece o primeiro número não positivo.

```c
#include <stdio.h>

int main()
{
    // A variável total é um acumulador
    float total=0, valor;

    while(1)    // laço de repetição infinito...
    {
        printf("Informe o valor do item: ");
        scanf("%f%*c", &valor);

        if(valor <= 0)
            break;  // termina o laço de repetição

        total += valor;
    }

    printf("\nTotal: R$ %.2f\n", total);

    return 0;
}
```

Figura 23 - Programa em C

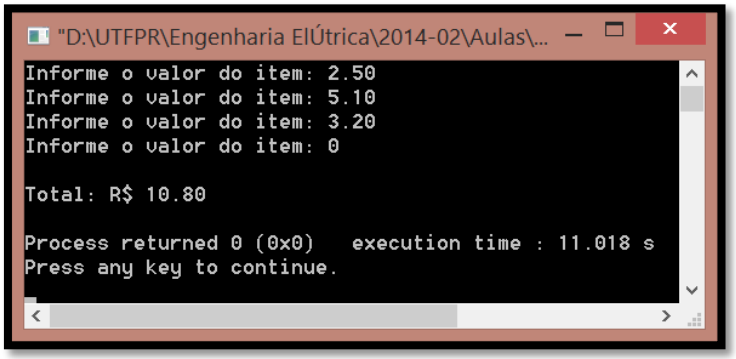
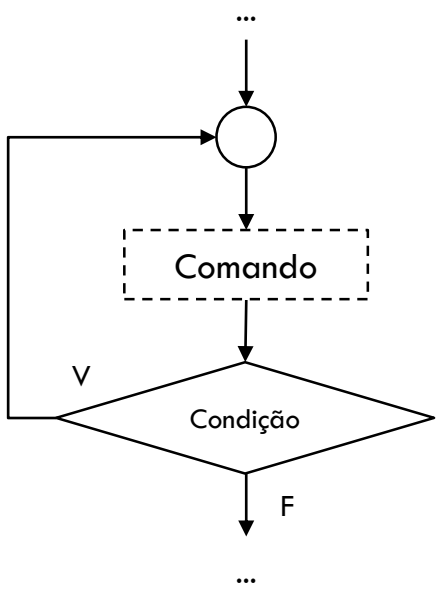


Figura 24 - Programa em execução

compras.c

ESTRUTURA DE REPETIÇÃO COM POSCONDIÇÃO

A **estrutura de repetição com poscondição** serve para executar um ou vários comandos, repetidamente, até que uma determinada condição se torne **falsa**. Essa estrutura é usada quando não sabemos de antemão quantas vezes o comando deve ser repetido, mas precisamos que ele seja executado pelo menos uma vez. Note que para executar mais de um comando, é preciso usar um par de chaves.




```
do
    comando;
while (condição);
```

```
do
{
    comando-1;
     ...
    comando-n;
} while (condição);
```

Padrão de codificação da estrutura de repetição com poscondição em linguagem C.

Em geral, a estrutura de repetição com poscondição é usada para:
- Garantir consistência de entrada de dados.
- Repetir um processo com confirmação do usuário.
- Implementar processos orientados por menus.

CONSISTÊNCIA DE ENTRADA DE DADOS

Para executar corretamente, os programas precisam que o usuário forneça dados **consistentes**. Por exemplo, um programa que calcula a raiz quadrada de um número fornecido pelo usuário não pode executar corretamente se esse número for negativo, pois como não existe raiz quadrada de números negativos, sua execução terminará com um **erro fatal** (ou seja, será abortada pelo sistema). Neste caso, em vez de aceitar qualquer valor digitado pelo usuário, o programa deve repetir a entrada de dados até que seja *consistente*.

CONSISTÊNCIA DE ENTRADA DE DADOS

Problema 7: Dado um número real não negativo, informe sua raiz quadrada.

```
Digite um número não negativo: –5 ↵
Digite um número não negativo: –17 ↵
Digite um número não negativo: 9 ↵
A raiz quadrada de 9.0 é 3.0
```

Figura 25 - Tela de execução

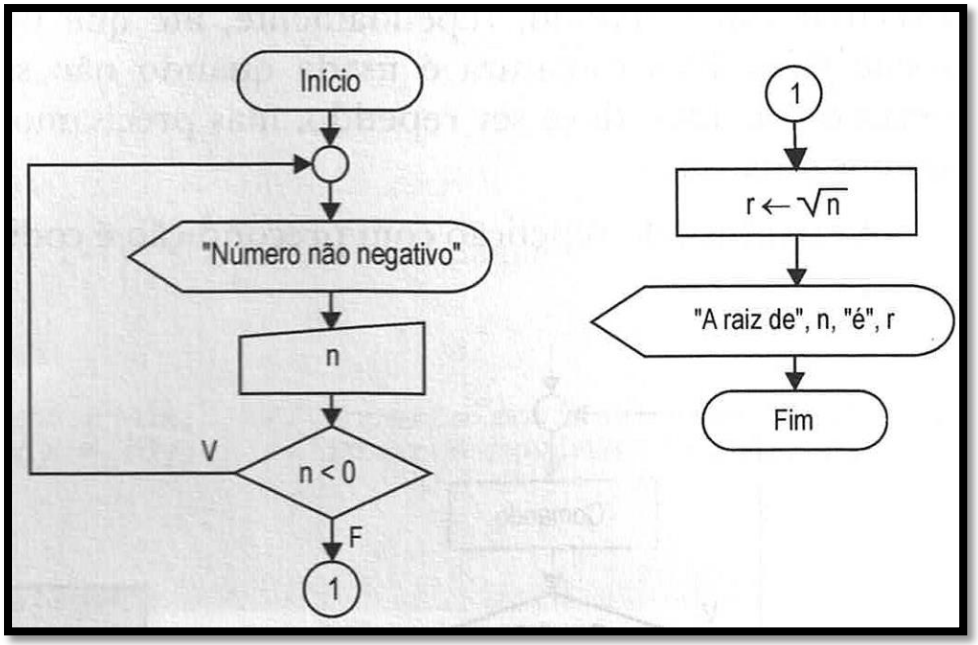


Figura 26 - Fluxograma

CONSISTÊNCIA DE ENTRADA DE DADOS

Problema 7: Dado um número real não negativo, informe sua raiz quadrada.

```c
/* Dado um número real não negativo, informe sua raiz quadrada. */

#include <stdio.h>
#include <math.h>

int main()
{
    // A variável total é um acumulador
    float numero, raiz_quadrada;

    // continua executando os comandos abaixo até que a
    // pessoa informe um número positivo
    do
    {
        printf("Informe um numero nao negativo: ");
        scanf("%f%*c", &numero);
    } while(numero < 0);

    raiz_quadrada = sqrt(numero);
    printf("\nRaiz Quadrada de %.2f: %.2f\n", numero, raiz_quadrada);

    return 0;
}
```

Figura 27 - Programa em C

raiz_quadrada.c

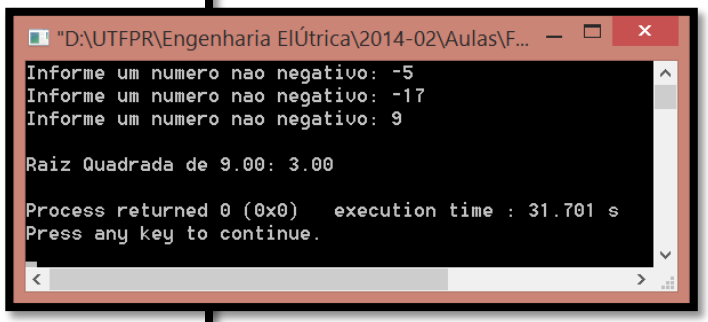


Figura 28 - Programa em execução

CONSISTÊNCIA DE ENTRADA DE DADOS

Problema 8: Dado um número inteiro entre 1 e 10, exiba a sua tabuada.

```
Número entre 1 e 10? –6 ↵
Número entre 1 e 10? 18 ↵
Número entre 1 e 10? 5 ↵
Tabuada do 5
5 x  1 =  5
5 x  2 = 10
5 x  3 = 15
...
```

Figura 29 - Tela de execução

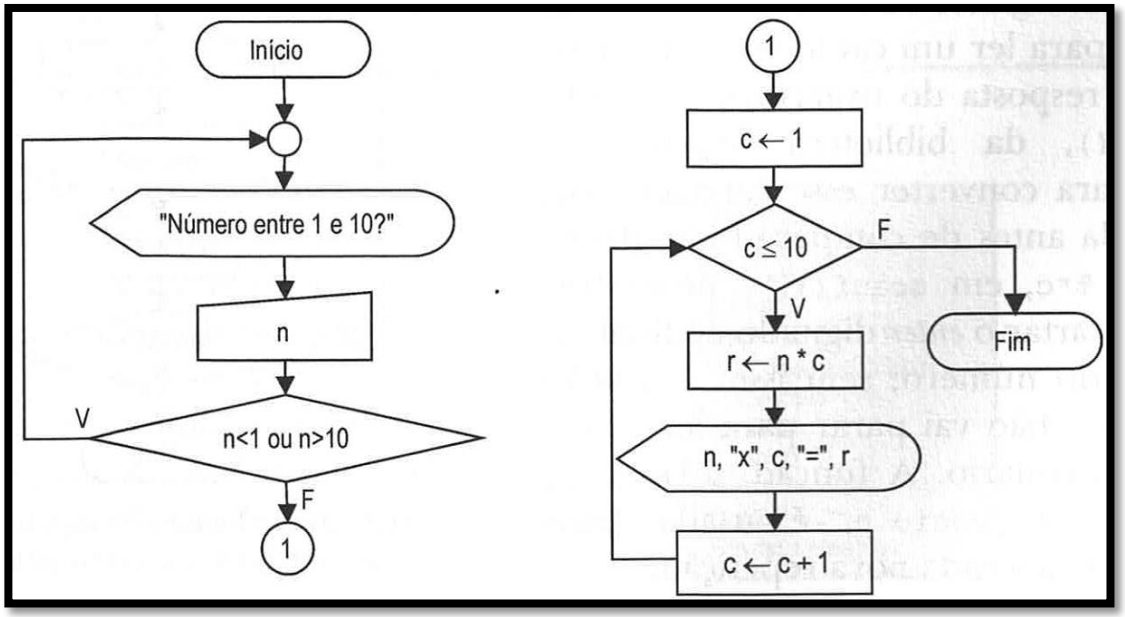


Figura 30 - Fluxograma

CONSISTÊNCIA DE ENTRADA DE DADOS

Problema 8: Dado um número inteiro entre 1 e 10, exiba a sua tabuada.

```c
/* Dado um número inteiro entre 1 e 10, exiba a sua tabuada. */

#include <stdio.h>

int main()
{
    int numero, contador, resultado;

    do
    {
        printf("Digite um numero entre 1 e 10: ");
        scanf("%d%*c", &numero);
    } while(numero < 1 || numero > 10);

    printf("\n****** TABUADA DO NUMERO %d ******\n\n", numero);

    for(contador=1; contador<=10; contador++)
    {
        resultado = numero * contador;
        printf("%d * %2d = %3d\n", numero, contador, resultado);
    }

    return 0;
}
```

Figura 31 - Programa em C

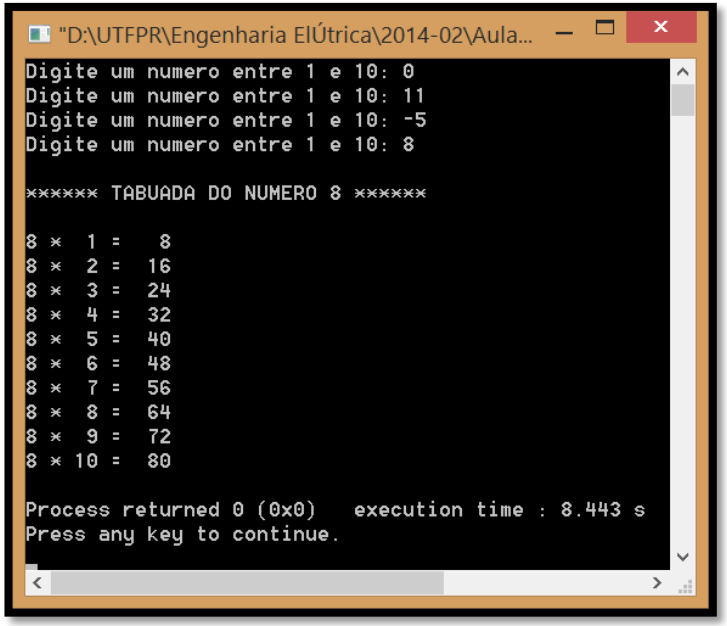


Figura 32 - Programa em execução

tabuada_com_consistencia_de_dados.c

REPETIÇÃO COM CONFIRMAÇÃO DO USUÁRIO

A **repetição com confirmação do usuário** consiste em um padrão em que um processo é executado e, ao final, o usuário é indagado se deseja continuar ou não a execução do mesmo.

O programa apresentado no próximo slide mostra o uso deste padrão. Nesse programa, a função **getchar()**, da biblioteca **stdio.h**, é usada para ler um caractere, representando a resposta do usuário, e a função **toupper()**, da biblioteca **ctype.h**, é usada para converter esse caractere em maiúscula antes de compará-lo a 'N'. O formato %*c, em **scanf()**, é necessário para descartar o ENTER digitado ao final da entrada do número; sem isso, a função **getchar()** não vai parar para ler a resposta do usuário. A função **system()** da biblioteca **stdlib.h** é utilizada para executar um comando do sistema operacional via terminal. O comando **CLS** dentro da função system faz com que a tela seja "limpada" a cada execução do programa.

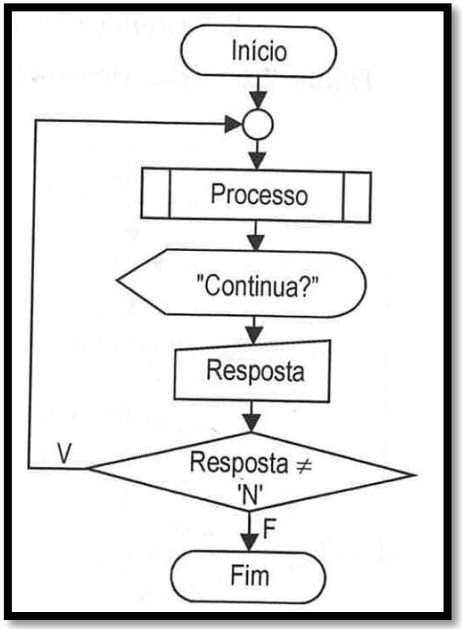


Figura 32 - Padrão de repetição com confirmação do usuário

REPETIÇÃO COM CONFIRMAÇÃO DO USUÁRIO

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

int main()
{
    int numero, contador, resultado;
    char opcao;

    do
    {
        system("CLS");

        do
        {
            printf("Digite um numero entre 1 e 10: ");
            scanf("%d%*c", &numero);
        } while(numero < 1 || numero > 10);

        printf("\n****** TABUADA DO NUMERO %d ******\n\n", numero);

        for(contador=1; contador<=10; contador++)
        {
            resultado = numero * contador;
            printf("%d * %2d = %3d\n", numero, contador, resultado);
        }

        printf("\nContinua (S/N)? ");
        opcao = toupper(getchar());

    } while(opcao != 'N');

    return 0;
}
```

Figura 33 - Programa em C

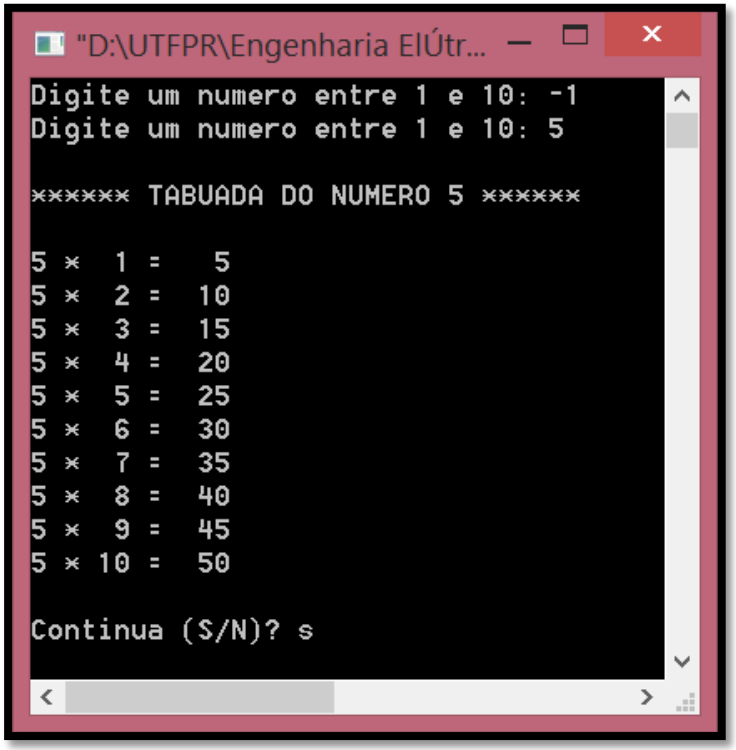


Figura 34 - Programa em execução

tabuada_com_confirmacao.c

PROCESSOS ORIENTADOS POR MENUS

Em um **processo orientado por menu**, uma série de opções é apresentada e, conforme a opção escolhida pelo usuário, uma ação correspondente é executada. Em seguida, o menu de opções é reapresentado e o processo se repete até que o usuário decida finalizar a sua execução.

Problema 9: Simular o funcionamento de um caixa eletrônico, que oferece as seguintes opções ao cliente: 1-depósito, 2-saque, 3-saldo e 4-sair. Suponha que o saldo inicial do cliente seja de R$ 1.000,00 e que o saldo pode ficar negativo.

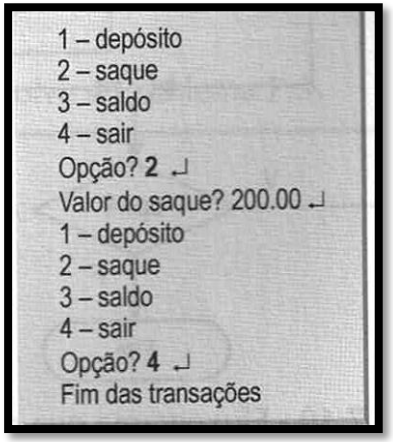


Figura 35 - Tela de execução

O programa apresentado no próximo slide usa a função **puts()** para exibir as opções do menu. Essa função, que faz parte da biblioteca **stdio.h**, é especializada para a exibição de cadeia de caracteres (*strings*) e tem a vantagem de mudar o cursor para uma nova linha, automaticamente, depois de exibir o texto. Evidentemente, o mesmo efeito pode ser obtido com o uso da função **printf()**, porém com **puts()** a sintaxe fica mais simples. Vale ressaltar que a função **puts()** só serve para a exibição de texto, não podendo ser usada para a exibição de valores de outros tipos de dados (**char**, **int** e **float**).

PROCESSOS ORIENTADOS POR MENUS

```c
/* Programa que simula o funcionamento de um caixa eletrônico */

#include <stdio.h>

int main()
{
    float saldo = 1000, valor;
    int opcao;

    do
    {
        puts("\n1 - Deposito");
        puts("2 - Saque");
        puts("3 - Saldo");
        puts("4 - Sair");
        printf("\nOpcao: ");
        scanf("%d%*c", &opcao);

        switch(opcao)
        {
            case 1:  printf("\nValor do deposito em R$: ");
                     scanf("%f%*c", &valor);
                     saldo += valor;
                     break;
            case 2:  printf("\nValor do saque em R$: ");
                     scanf("%f%*c", &valor);
                     saldo -= valor;
                     break;
            case 3:  printf("\nSaldo atual = R$: %.2f\n", saldo);
                     break;
            default: if(opcao != 4)
                        puts("Opcao invalida!");

        }
    } while(opcao != 4);

    puts("\nObrigado por utilizar os nossos servicos :D");

    return 0;
}
```

Figura 36 - Programa em C

Figura 37 - Programa em execução

caixa_eletronico.c

EXERCÍCIOS

exercicio1_slide_32.c

exercicio3_slide_32.c

exercicio2_slide_32.c

1. Dadas as duas notas de um aluno, informe a sua média. Seu programa deve forçar o usuário a digitar notas na faixa de 0 a 10.

2. Modifique a lógica elaborada para o exercício anterior de modo que, ao final de cada execução, o usuário tenha a opção de repetir o processo.

3. Crie um programa que exiba um menu com as seguintes opções:

    1 - somar

    2 - subtrair

    3 - multiplicar

    4 - dividir

    5 - sair

Após a escolha da opção, o usuário deve fornecer dois números e o programa deve mostrar o resultado da operação.

4. Escolha um programa que você tenha feito anteriormente e altere-o de modo que, ao final, o usuário tenha a opção de repetir o processo.

RESUMO

contar++ significa aumentar contar em 1.

contar-- significa diminuir contar em 1.

Loops FOR são uma maneira mais compacta de escrever loops.

Loops WHILE repetem o código, contanto que a condição seja verdadeira.

Loops DO-WHILE executam o código pelo menos uma vez.

Os laços de repetição são conhecidos também por sua tradução em inglês: *loops* ou *looping*.

REFERÊNCIAS BIBLIOGRÁFICAS

- ASCENCIO, A. F. G.; CAMPOS, E. A. V. D. **Fundamentos da Programação de Computadores: Algoritmos, Pascal, C/C++ (Padrão ANSI) e Java**. 3. ed. São Paulo: Pearson Education do Brasil, 2012. 569 p.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação: A construção de algoritmos e estruturas de dados.** 3. ed. São Paulo: Prentice Hall, 2005. 218p.
- GRIFFITHS, D. **Use a Cabeça - C**. 1. ed. São Paulo: Alta Books: 2013. 632p.
- PEREIRA, S. D. L. **Algoritmos e Lógica de Programação em C: Uma abordagem didática**. 1. ed. São Paulo: Érica, 2010. 190 p.