

UNIDADE III
CONTROLE DE FLUXO AVANÇADO

Prof. Esp. Carlos Danilo Luz
Prof. Esp. João Messias Pereira Lenco

Objetivos de Aprendizagem

Para essa unidade, os objetivos concentram-se em apresentar os conceitos de laços de repetição; mostrar como deve-se utilizar efetivamente esse tipo de estrutura; além de possibilitar a você, aluno(a), conhecer a sintaxe para utilização da estrutura de repetição; ainda faz parte da proposta dessa unidade, compreender como ocorre a execução de uma estrutura de repetição; e, por fim, iremos estudar as estruturas de repetição com base em operadores lógicos.

Plano de Estudo

Nesta unidade, serão abordados os seguintes tópicos:

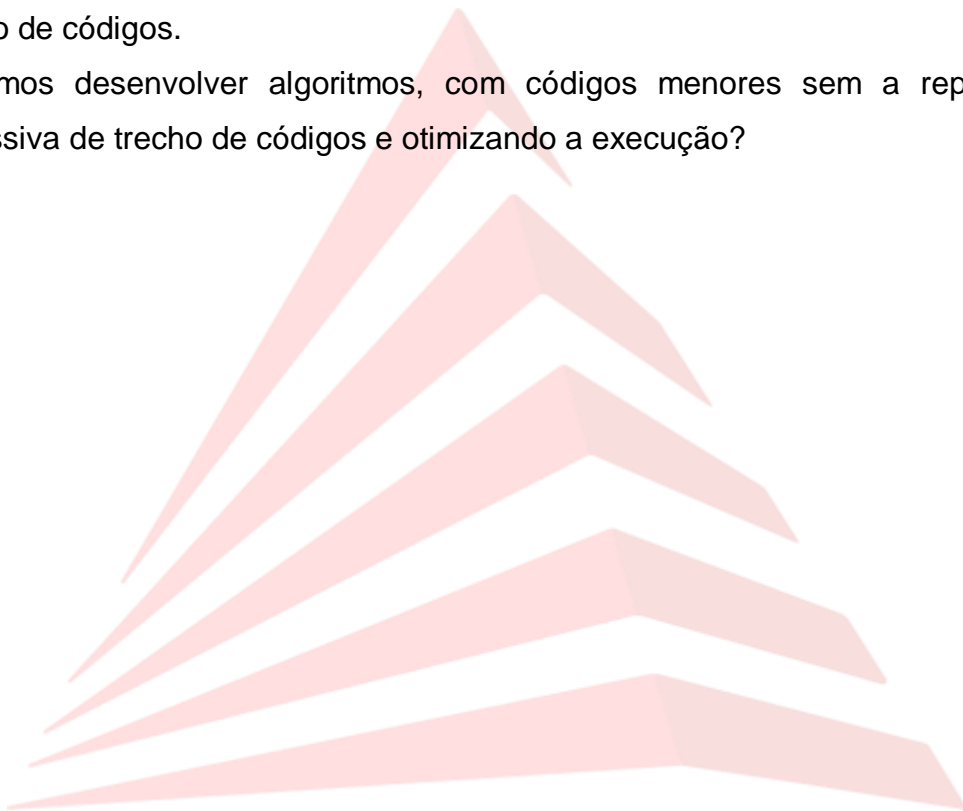
1. Operador de Seleção *Switch*
2. Conceitos Básicos de Repetição
3. Estrutura de Repetição *While* e *do/While*
4. Exemplos Dirigidos

CONVERSA INICIAL

Os programas que utilizamos possuem características por vezes únicas com funções, procedimentos, requisitos. Eventualmente, temos situações que exigem uma entrada de dados um pouco grande, isso faz com que tenhamos que replicar por diversas vezes um determinado trecho de código, geralmente quando efetuarmos a entrada de dados.

Uma solução deste fato, independente da linguagem de programação a ser desenvolvido o código, é utilizar estruturas que auxiliam em repetições de trecho de códigos.

Podemos desenvolver algoritmos, com códigos menores sem a repetição excessiva de trecho de códigos e otimizando a execução?



1.0 OPERADOR DE SELEÇÃO SWITCH

As estruturas condicionais *IF* e *IF - ELSE* são úteis em casos de tomadas de decisões simples, nos quais temos apenas dois desvios a serem tomados, com base nas expressões ou operadores lógicos. Entretanto, podemos nos deparar com situações em que temos diversos caminhos a serem seguidos, podemos chamar isto de seleção múltipla.

[...] C tem um comando interno de seleção múltipla, *switch*, que testa sucessivamente o valor de uma expressão contra uma lista de constantes internas ou de caractere. Quando o valor coincide, os comandos associados àquela constante são executados. (SCHILDT, 1997 p. 70) [...]

Podemos dizer que a estrutura de seleção *switch* é uma evolução das estruturas condicionais simples e compostas, pois, por meio desta, podemos ter um leque de opções e desvios.

1.1 Estrutura do *switch*

A estrutura de seleção múltipla *switch* deixa os algoritmos mais dinâmicos, pois podemos testar diversas condições a partir de uma variável. Vamos pensar no seguinte cenário: um sistema de cadastro de indicação de filmes de uma locadora, com base em classificações indicativas¹, ao desenvolvermos um algoritmo que efetue a leitura do dado informado, podemos ter diversas opções para a indicação.

Na linguagem C, a estrutura *switch* é utilizada em conjunto com a expressão *case*, podemos compreender sendo uma lista com a validação dos desvios. A execução da estrutura *switch* compara todos os *cases* até encontrar um desvio condicional satisfatório, assim executando seu bloco de código, após iniciar a sequência de códigos, chamamos o comando *break*, o qual veremos com mais detalhes no próximo tópico.

Caso nenhum dos itens do *switch* seja atendido, a estrutura *default* é acionada, este item é identificado sendo a saída padrão, caso nenhum dos *case* atenda à

¹ ESCLARECIMENTO: As classificações indicativas são regulamentações sobre as idades indicadas para consumir cada obra. Elas são feitas pela Secretaria Nacional de Justiça (SNJ), do Ministério da Justiça brasileiro. <http://cinematecando.com.br/conheca-as-classificacoes-indicativas/>

necessidade, neste momento, o bloco de código a ser executado será apenas o *default*. Vejamos, a seguir, na figura “Composição do SWITCH CASE textual”, os comandos em linguagem C:

Figura 48. Composição do SWITCH CASE textual

```
switch (expressão){  
  
    case condição1:  
        <bloco de códigos>  
        break;  
  
    case condição2:  
        <bloco de códigos>  
        break;  
  
    case condição3:  
        <bloco de códigos>  
        break  
    .  
    .  
    .  
    default:  
        <bloco de código padrão>  
}  

```

Fonte: Os autores (2019).

Mesmo sendo uma estrutura mais poderosa que as de IF - ELSE, devemos atentar para alguns pontos importantes. Segundo Schildt (1997 p. 70-71), estes itens são:

1. O comando *switch* difere do comando *if* porque *switch* só pode testar igualdade, enquanto *if* pode avaliar uma expressão lógica ou relacional.
2. Duas constantes *cases* no mesmo *switch* não podem ter valores idênticos. Obviamente, um comando *switch*, incluído em outro *switch* mais externo, pode ter as mesmas constantes *case*.
3. Se constantes de caractere são usadas em um comando *switch*, elas são automaticamente convertidas para seus valores inteiros.

Para compreendermos melhor, vamos relembrar o cenário anteriormente apresentado, no qual estamos desenvolvendo um sistema de cadastro de

indicação de filmes com base na informação gerada do usuário e a classificação indicativa. Vejamos o código-fonte em C, a seguir, na figura “Algoritmo de classificação de filmes”.

Figura 49. Algoritmo de classificação de filmes

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

main (){
    setlocale(LC_ALL, "Portuguese");
    int classificacao;

    printf("-----\n");
    printf("0 = Livre \n1 = 10 anos \n");
    printf("2 = 12 anos \n3 = 14 anos \n");
    printf("4 = 16 anos \n5 = 18 anos \n");
    printf("9 = Sem Classificação \n");
    printf("-----\n");
    printf("Digite a classificação indicativa do filme: ");
    scanf("%d", &classificacao);

    switch (classificacao){
        case 0 :
            printf("\n Classificação livre: desenho animados.\n");
            break;

        case 1 :
            printf("\n Classificação 10: presença de armas, medo e
tensão.\n");
            break;

        case 2 :
            printf("\n Classificação 12: violência, lesão corporal,
descrição de violência.\n");
            break;

        case 3 :
            printf("\n Classificação 14: morte, vulgaridade, drogas
ilícitas.\n");
            break;

        case 4 :
            printf("\n Classificação 16: tortura, mutilação, violência,
morte.\n");
            break;
```

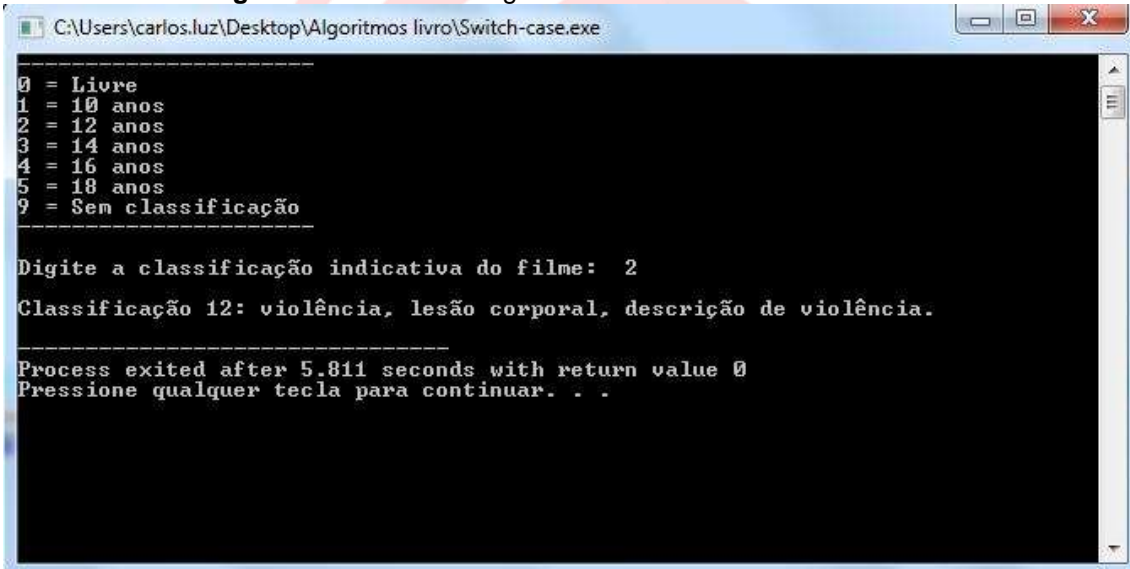
```
        case 5 :  
            printf("\n Classificação 18: violência de forte impacto,  
crueldade.\n");  
            break;  
        default:  
            printf("\n Sem classificação.\n");  
    }  
    return(0);  
}
```

Fonte: os autores (2019, programa utilizado DEV C++).

No algoritmo apresentado, será exibida na tela a classificação indicada, conforme os dados informados pelo usuário, a estrutura será percorrida até encontrar uma condição satisfatória ou será apresentada a mensagem que consta no desvio *default*.

Vejamos este algoritmo em execução na figura “Retorno do algoritmo de adivinhe o número oculto”.

Figura 50. Retorno do algoritmo de adivinhe o número oculto



```
C:\Users\carlos.luz\Desktop\Algoritmos livro\Switch-case.exe  
0 = Livre  
1 = 10 anos  
2 = 12 anos  
3 = 14 anos  
4 = 16 anos  
5 = 18 anos  
9 = Sem classificação  
-----  
Digite a classificação indicativa do filme: 2  
Classificação 12: violência, lesão corporal, descrição de violência.  
-----  
Process exited after 5.811 seconds with return value 0  
Pressione qualquer tecla para continuar. . .
```

Fonte: os autores (2019, programa utilizado DEV C++).

Ao executar o algoritmo, digitamos o número 2, que equivale à classificação de 12 anos, quando a estrutura processada é comparada ao primeiro case, como este não é o número 2 e passado para o segundo case este também não corresponde ao número 2, já o terceiro case corresponde com o que foi

digitado, neste momento, o que estiver associado a ele será executado, com base em nosso algoritmo a informação exibida em tela terá sido “Classificação 12...”; compreender esta estrutura é interessante, porque funciona praticamente com a operação lógica de comparação, igualdade de elementos, não sendo aplicada quando trabalhamos com outros operadores lógicos como os de maior que, menor que, maior ou igual, dentre outros.

1.2 Comando *break*

O comando *break* é implementado em diversas estruturas, uma destas é a *switch* - case que acabamos de ver no tópico anterior, podemos utilizar o comando *break* ou o comando *continue*, para representar a parada ou o andamento da execução do bloco de códigos a serem executados.

Os comandos *break* e *continue* são usados para alterar o fluxo de controle. O comando *break*, quando executado em uma estrutura *while*, *for*, *do...while* ou *switch* causa uma saída imediata dessa estrutura. A execução do programa continua com a próxima instrução. Os usos comuns do comando *break* são para escapar mais cedo de um loop ou para pular o restante de uma estrutura *switch*. (DEITEL; DEITEL, 2011 p. 94)

Sobre as estruturas mencionadas, *while* - *for* - *do...while*, estas são consideradas como estruturas de repetição ou laço de repetição para compreendermos o quão importante é o uso do comando *break* na estrutura *switch*, vejamos o código, a seguir, sem o uso do comando *break* na figura “Algoritmo de classificação de filmes”.

Figura 51. Algoritmo de classificação de filmes

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

main (){
    setlocale(LC_ALL, "Portuguese");
    char operacao;

    printf("Escolha uma oeração [ + - * / ]:");
    scanf("%c", &operacao);

    switch (operacao){
        case '+':
            printf("\n Adição.\n");
            break;

        case '-':
            printf("\nSubtração");
            break;

        case '*':
            printf("\nMultitplicação");
            break;

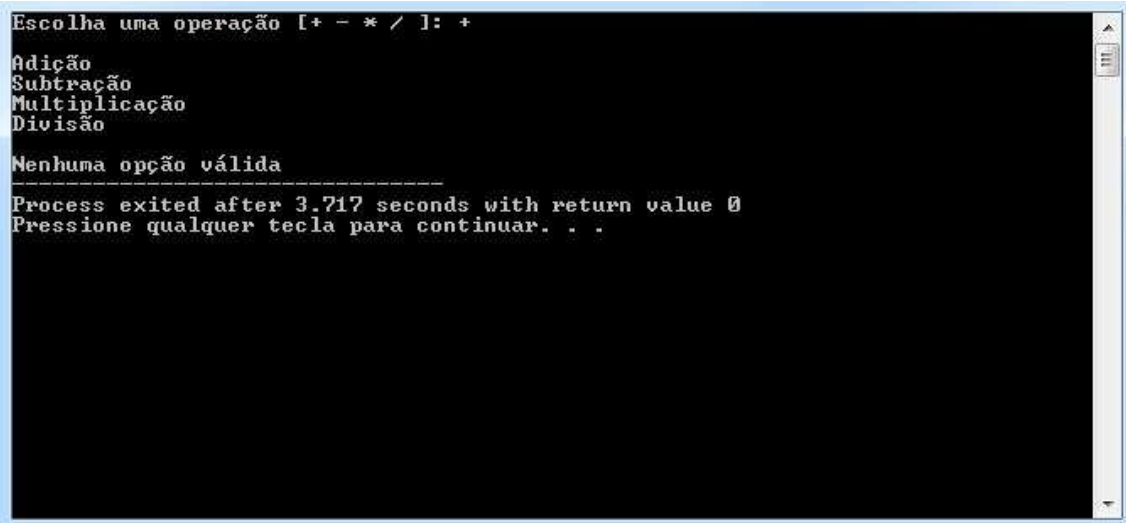
        case '/':
            printf("\nDivisão");
            break;

        default:
            printf("\nNenhuma operação válida");
    }
    return(0);
}
```

Fonte: os autores (2019, programa utilizado DEV C++).

Este algoritmo é bem simples, ele requisita para o usuário escolher uma operação aritmética, após isso, executamos a estrutura *switch - case*, observe que neste código-fonte desenvolvido em linguagem C não utilizamos o comando *break* após a comparação de cada *case*, neste caso, vejamos este algoritmo em execução na figura “Retorno do algoritmo sem o comando *break*”.

Figura 52. Retorno do algoritmo sem o comando break



```
Escolha uma operação [+ - * / ]: +
Adição
Subtração
Multiplicação
Divisão

Nenhuma opção válida
-----
Process exited after 3.717 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Fonte: os autores (2019, programa utilizado DEV C++).

Ao executar o algoritmo e digitarmos o sinal de adição a estrutura *switch* é processada, note que mesmo tendo um caminho de desvio compatível, todos os cases foram acessados e impressos na tela. O comando *break* é necessário nesta e em outras estruturas para dar a saída da estrutura e não executar o bloco de códigos errados, com base nisto, vemos que é sempre necessário utilizar o comando *break* na estrutura de seleção múltipla.

1.3 Exemplo: imprimindo números na forma literal

Para compreendermos melhor o funcionamento da estrutura de seleção múltipla, *switch - case*, vamos desenvolver um algoritmo no qual o usuário digite um número entre 0 e 9, após isso, esse dado de entrada será processado pela estrutura *switch* e impresso na tela qual o número escolhido de forma literal. Vejamos o código-fonte desenvolvido em linguagem C, a seguir, na figura “Algoritmo de número digitado e impresso de forma literal”.

Figura 53. Algoritmo de número digitado e impresso de forma literal

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

int numero;

main(){
    setlocale(LC_ALL, "Portuguese");

    printf("Escolha um número de 0 à 9: ");
    scanf("%d",&numero);

    switch (numero){
        case 0:
            printf("Número digitado: ZERO");
            break;
        case 1:
            printf("Número digitado: UM");
            break;
        case 2:
            printf("Número digitado: DOIS");
            break;
        case 3:
            printf("Número digitado: TRÊS");
            break;
        case 4:
            printf("Número digitado: QUATRO");
            break;
        case 5:
            printf("Número digitado: CINCO");
            break;
        case 6:
            printf("Número digitado: SEIS");
            break;
        case 7:
            printf("Número digitado: SETE");
            break;
        case 8:
            printf("Número digitado: OITO");
            break;
        case 9:
            printf("Número digitado: NOVE");
            break;
        default:
            printf("\n\nNúmero digitado errado!");
    }
}
```

```
}  
  
}
```

Fonte: os autores (2019, programa utilizado DEV C++).

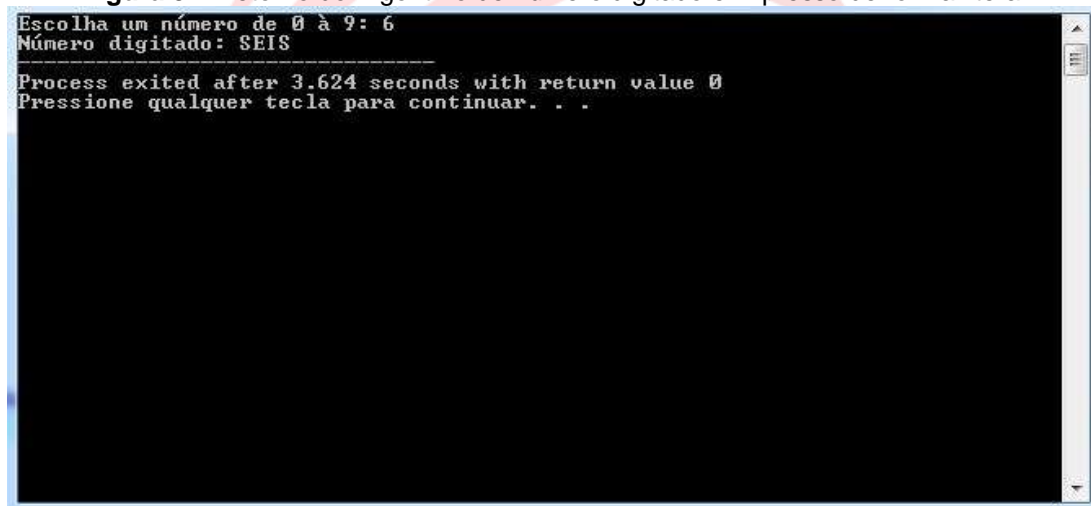
Reflita

Sendo a linguagem C uma das precursoras de linguagens mais recentes, muitos de seus conceitos podem ser aplicados em outras linguagens?

Nota-se que todo o tratamento da informação será realizado pela estrutura de seleção *switch*, a qual irá comparar o valor digitado pelo usuário com cada desvio os case. Caso algum desses desvios seja compatível com o número digitado, este será acessado e impresso na tela, materializando a informação de qual foi o número escolhido de modo literal.

Vejamos o algoritmo em execução na figura “Retorno do Algoritmo de número digitado e impresso de forma literal”, a seguir.

Figura 54. Retorno do Algoritmo de número digitado e impresso de forma literal



```
Escolha um número de 0 à 9: 6  
Número digitado: SEIS  
-----  
Process exited after 3.624 seconds with return value 0  
Pressione qualquer tecla para continuar. . .
```

Fonte: os autores (2019, programa utilizado DEV C++).

Ao executar o algoritmo e ser digitado o número seis, o algoritmo executa a estrutura de seleção *switch* até encontrar, no caso, foi apresentado na tela “Número digitado: SEIS”. Mas, como apresentado anteriormente, se o usuário tivesse digitado o número 11, nenhuma das situações seria atendida, sendo,

então, acionado e impresso na tela o conteúdo que consta dentro do desvio *default*, caracterizando que nenhum dos *cases* atendeu à comparação.

A partir do exemplo apresentado com a estrutura *switch*, podemos desenvolver diversas variações deste mesmo algoritmo, deixando, dessa forma, cada vez mais completos e complexos seus algoritmos desenvolvidos.

Indicação De Recurso Didático



Algoritmos e lógica de programação

Ricardo Concílio (Autor), Marcelo Gomes (Autor), Marcio Soares (Autor), Marco Souza (Autor)

Descrição da obra: Com linguagem simples e didática, o livro procura tornar a lógica de programação prática, além de mostrar aos estudantes um caminho mais adequado na construção dos algoritmos. A abstração de procedimentos e dados é um dos maiores problemas para os estudantes nos cursos introdutórios, e, para tentar escapar das dificuldades, os autores utilizam uma arquitetura de computador simples, baseada na arquitetura de Von Neumann, de maneira a fixar os conceitos relacionados à operação de computadores. Um dos principais objetivos do livro é fazer que o estudante consiga no futuro relacionar os aspectos abstratos da computação com sua implementação, e ainda incentivar a necessidade de escrever os algoritmos antes de sua implementação propriamente dita. A descrição dos algoritmos no texto é mostrada por meio de fluxogramas

2. CONCEITOS BÁSICOS DE REPETIÇÃO

Uma das vantagens encontradas na computação é capacidade dos computadores em realizar determinadas tarefas repetidas vezes em um curto espaço de tempo, efetuadas em frações de segundos. De fato, algo muito rápido e quase imperceptível para o cérebro humano.

Ao escrever um programa em linguagem C, existem muitas situações em que precisamos repetir um determinado trecho de código por inúmeras vezes. Um exemplo simples disso é quando queremos identificar a quantidade de números pares existentes em um intervalo de 0 a 100, ou, então, quando queremos que o sistema continue aberto até que o usuário digite a palavra “SAIR”. Para situações como esta podemos utilizar uma estrutura que permite estabelecer uma série de repetições do trecho de código desejado, quantas vezes forem necessárias. Uma estrutura de repetição também pode ser definida como um laço de repetição ou malhas de repetição. E para a série de repetições utilizamos o termo *loop*.

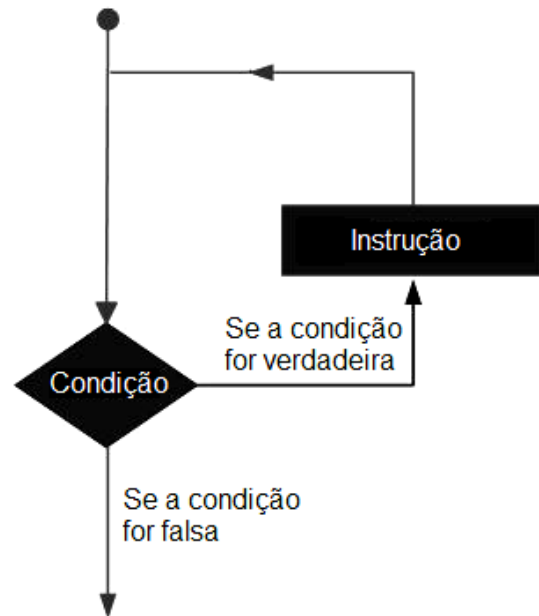
2.1 A Lógica do Loop

O *loop* é uma das principais características da programação estruturada, pois permite que a tomada de decisão aconteça a partir de uma determinada condição. Basicamente, um *loop* pode ter seu número de repetições definido de maneira fixa ou condicional. Na linguagem C existem 3 tipos de estruturas de repetição: *for*, *while*² e *do while*. O *for* é uma estrutura do tipo laço contado, pois utiliza um valor previamente definido para a quantidade de repetições que irá executar. Já as estruturas *while* e *do while* representam laços condicionais, pois o número de repetições está relacionado a uma determinada condição ser atendida ou não.

De maneira gráfica, podemos exemplificar o funcionamento de um *loop* da seguinte forma:

² Curiosidade: em português ou português estruturado, o *For* e o *While*, correspondem aos comandos Para e Enquanto, respectivamente.

Figura 55. Exemplo gráfico de loop.



Fonte: Os autores (2019).

Sempre que a condição for verdadeira o programa irá executar um fluxo alternativo, seguindo as instruções previamente definidas para este caso, garantindo assim que alguma ação será executada. No entanto, caso a condição seja falsa, o programa irá seguir o fluxo normal do algoritmo, caracterizando a saída do *loop*.

Exemplo utilizando o comando *for*:

Figura 56. Exemplo de repetição utilizando o *for*.

```
#include<stdio.h>

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

Fonte: Os autores (2019).

Esta estrutura estabelece que a variável X inicia seu valor em 0 ($X=0$), porém, enquanto seu valor for menor ou igual a 10, X irá receber um incremento de 1 ($X++$) e, em seguida, o valor de X deverá ser impresso na tela.

Segue abaixo o resultado desta execução:

Figura 57. Exemplo utilizando *for* compilado.

```
0
1
2
3
4
5
6
7
8
9
10
-----
Process exited after 0.5424 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Fonte: Os autores (2019).

Neste exemplo, foi utilizado um elemento contador ($++$), que veremos a seguir nesta unidade.

2.2 Um *Loop* Infinito

Uma estrutura de repetição é caracterizada por sua capacidade de executar uma determinada instrução ou um grupo de instruções repetidas vezes até que uma determinada condição seja satisfeita. Esta condição é responsável por definir o elemento de parada do *loop*, sendo, então, fundamental para que o número de repetições desejada aconteça da maneira correta. Quando esse elemento não é informado da maneira adequada ou simplesmente não é informado, o programa estará sujeito à ocorrência de um *loop infinito*. Isto acontece sempre que uma condição nunca se torna falsa ou que seu elemento de parada nunca é alcançado ou satisfeito.

Exemplo utilizando o comando *for*:

Figura 58. Loop infinito.

```
#include<stdio.h>

int main()
{
    int x;
    for(x=0; ; x++)
        printf("%d", x);
}
```

Fonte: Os autores (2019).

Neste exemplo, note que não existe uma condição de parada, ou seja, até quando o laço deve ser executado. Dessa forma, ao executar este programa, a impressão da variável X acontecerá de maneira infinita.

2.3 Repetição com Contadores, *Flags* e Valores de Entrada

Antes de nos aprofundarmos um pouco mais nas estruturas de repetição *for*, *while* e *do while* é importante ressaltar algumas das características que compõem estas estruturas. Como já dito anteriormente, os laços de repetição podem ser definidos com elementos ou condições de parada que encerram sua execução. Quando não é informado nenhum elemento deste tipo ou define-se uma condição que nunca será alcançada, temos o que é chamado de *loop infinito*.

As estruturas de repetição podem ser definidas com elementos contadores, *flags* ou valores de entrada, que são responsáveis por auxiliar a condição de parada. Estes três elementos são definidos da seguinte forma:

Contadores (++ / --): permite que o laço de repetição faça o incremento ou decremento de um valor a cada iteração³. Assim, enquanto a condição do laço não for satisfeita, o contador fica encarregado de atualizar o valor da variável que está sendo verificada dentro do laço.

Flags: funcionam como uma espécie de dado falso, o qual jamais será alcançado dentro da estrutura de repetição. Geralmente, são utilizadas quando

³Esclarecimento: Iterações corresponde a cada ciclo que um laço de repetição executa, isto é, cada vez que o valor da variável é verificado dentro da condição.

não se sabe ao certo o número total de uma variável, o que de certa forma ocasiona o *loop* infinito de maneira proposital, fazendo com que a verificação seja feita infinitamente. Um exemplo simples é quando queremos calcular a média de idade de um determinado grupo de funcionários, mas não se sabe ao certo quantos funcionários serão consultados. Sendo assim, define-se uma idade falsa para que seja verificada, por exemplo, 200 anos. Neste caso, o laço de repetição será executado até que um funcionário seja informado com a idade 200 anos, o que certamente não irá acontecer.

Valores de Entrada: os valores de entrada permitem estabelecer uma condição de parada a partir de uma informação digitada pelo usuário. Enquanto um valor previamente definido não é digitado pelo usuário, o programa continua a execução do laço de repetição. Uma situação bastante comum são os menus de cadastro de cliente, que permanecem ativos enquanto a palavra “sair”, por exemplo, não for digitada. Esse tipo de condição de parada torna-se uma alternativa para o caso das *Flags*. Assim, mesmo que uma estrutura esteja sendo executada com um número de repetições indefinida, podemos estabelecer um elemento de parada por meio dos valores de entrada.

Indicação De Recurso Didático



Aprenda Lógica de Programação e Algoritmos com Implementações em Portugol, C, Java, C# e Python

Cláudio Luís Vieira Oliveira (Autor)

Descrição: A programação de computadores tem se tornado cada vez mais fácil, acessível e popular, pois no mundo atual, o uso da tecnologia está fortemente inserido no cotidiano das pessoas, criando um universo de novas possibilidades. Este livro é o resultado das experiências adquiridas pelos autores ao longo de mais de uma década dedicada ao ensino nos cursos de graduação em Informática. No primeiro capítulo, o leitor irá encontrar os conceitos empregados para a resolução de problemas computacionais usando, para isso, técnicas e ferramentas como fluxograma, Portugol e a Scratch, que é uma linguagem de programação visual que possibilita aprendizado rápido e lúdico.

3.0 Loops Aninhados

Estruturas de repetição podem ser definidas individualmente ou também de forma aninhada, também chamada de estrutura de repetição encadeada. Isso ocorre quando temos uma estrutura dentro de outra, permitindo que sejam definidos subníveis de estruturas de repetição. Veja o exemplo:

Figura 59. Estrutura de repetição aninhada.

```
#include<stdio.h>

int main()
{
    int linha, coluna;

    linha = 1;
    int i, j;
    for (i=0; i<3; i++)
    {
        printf( "\nLinha %d \n", i);
        for (j=0; j<3; j++)
        {
            printf( "Coluna - %d ", j);
        }
        printf( "\n" );
    }
}
```

Fonte: Os autores (2019).

A estrutura apresentada no exemplo destaca o uso de uma estrutura *for* dentro de outra. Neste caso, vale ressaltar que as variáveis que serão utilizadas dentro da estrutura interna, precisam ser declaradas de maneira externa (global), para que assim possam ser acessadas sem que ocorram restrições de localização delas. Caso sejam declaradas de maneira interna (local), o programa pode não funcionar da maneira esperada, pois a estrutura externa também precisa ter acesso a estas variáveis. Ao executar este código, temos o seguinte resultado:

Figura 60. Estrutura de repetição aninhada impressa.

```
Linha 0
Coluna - 0 Coluna - 1 Coluna - 2
Linha 1
Coluna - 0 Coluna - 1 Coluna - 2
Linha 2
Coluna - 0 Coluna - 1 Coluna - 2
-----
Process exited after 0.4218 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Fonte: Os autores (2019).

Neste exemplo, a lógica apresentada é a seguinte: o primeiro *for* é responsável por criar uma linha de tabela enquanto o segundo *for* fica encarregado de criar as colunas para esta tabela. Em ambas as estruturas foram definidas um limite de, no máximo, três repetições, ou seja, para cada linha na tabela, teremos três colunas. No total, serão três linhas compostas de três colunas cada.

3.1 Estrutura de Repetição *while*

Uma estrutura de repetição ou também chamada de laço de repetição é utilizada quando temos que efetuar um mesmo procedimento por algumas vezes. Frequentemente os softwares precisam repetir a execução de um bloco de códigos até que determinada condição seja atendida, ou até uma determinada quantidade de vezes.

Devo salientar que a composição e a forma de execução de uma estrutura de repetição são praticamente as mesmas para as mais diversas linguagens, não sendo limitado apenas para a linguagem C, outras linguagens como JAVA, PHP, DELPHI, dentre outras, se utilizando dos laços de repetição seguindo a mesma ideia.

3.2 Estrutura do *while*

A estrutura de repetição é utilizada em diversas situações, por meio dela podemos especificar a quantidade de vezes que queremos que um determinado bloco de código é executado. Cada estrutura de repetição é uma particularidade, caso esta não seja atendida, o bloco de código pode ou não ser executado.

Uma das vantagens de se utilizar um laço de repetição é um código-fonte menor e mais fácil de se manter, pois reduzimos para poucas linhas o bloco de execução, cada estrutura de repetição tem a sua particularidade, mas todas têm o mesmo fim, a repetição, até que a condição a ser analisada seja satisfeita.

O comando `while` consiste na palavra-chave `while` seguida de uma expressão de teste entre parênteses. Se a expressão de teste for verdadeira, o corpo do laço é executado uma vez e a expressão de teste é avaliada novamente. Esse ciclo de teste e execução é repetido até que a expressão de teste se torne (igual a zero), então o laço termina e o controle do programa passa para a linha seguinte ao laço. (MIZRAHI, 2008 p. 73)

A particularidade da estrutura de repetição `while` está justamente no fato de que para executá-la, deve-se atender a uma determinada condição para assim ser executado o laço, a cada laço esta condição é testada. Vejamos a seguir a sua composição.

Figura 61. Composição da estrutura de repetição *While*

```
while (condição){  
    <bloco de código a ser executado>  
  
    função de incremento  
}
```

Fonte: Os autores (2019).

Para compreendermos melhor como podemos utilizar a estrutura `while`, vamos compreender melhor o algoritmo que foi implementado quando trabalhamos estudamos sobre `if` aninhados, o algoritmo no qual o usuário tenta descobrir qual o número oculto, a diferença, neste caso, é que ele não terá mais um limite de chances e o número está em torno de 0 a 100, em que o usuário irá contar com as dicas, se o número oculto é maior ou menor do que o digitado pelo usuário. Vejamos o código fonte a seguir.

Figura 62. Estrutura de repetição *while*

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

int main()
{
    setlocale(LC_ALL, "Portuguese");
    int nOculto, nUsuario = 0;
    nOculto = rand() % 100;

    printf("Descubra qual o numero oculto, entre 0 e 100. \nVocê não tem um limite de
chances\n\n");
    while(nUsuario != nOculto)
    {
        printf("Qual o seu primeiro palpite? ");
        scanf("%d", &nUsuario);
        // estrutura IF - ELSE
        if (nUsuario == nOculto){
            printf("Parabéns você descobriu o número %d \n", nOculto);
            break; // força a parada do algoritmo
        }else{
            if(nOculto > nUsuario){
                printf("O Número oculto é maior que %d \n\n", nUsuario);
            }else{
                printf("O Número oculto é menor que %d \n\n", nUsuario);
            }
        }
    }
}
```

Fonte: os autores (2019, programa utilizado DEV C++)

Note que a expressão no laço de repetição *while* é **nUsuario != nOculto**, sendo que se o número digitado pelo usuário foi diferente do número oculto, o laço de repetição ficará em constante *loop* até que o usuário digite o número correto, para que possamos forçar a entrada no laço de repetição *while*, já que este só se inicia se a expressão foi atendida. Testando a expressão no início, inicializamos a variável *nUsuario* sendo 0, desta forma, podemos considerar que o laço sempre vai ser executado, e mesmo que o número oculto seja realmente zero, o laço não será executado. Vejamos o algoritmo em execução.

Figura 63. Retorno do algoritmo “acerte o número oculto”

```
Descubra qual o número oculto, entre 0 e 100.  
Você não tem um limite de chances  
  
Qual o seu primeiro palpite? 36  
O Número oculto é maior que 36  
  
Qual o seu primeiro palpite? 50  
O Número oculto é menor que 50  
  
Qual o seu primeiro palpite? 40  
O Número oculto é maior que 40  
  
Qual o seu primeiro palpite? 45  
O Número oculto é menor que 45  
  
Qual o seu primeiro palpite? 41  
Parabéns você descobriu o número oculto 41  
  
-----  
Process exited after 17.92 seconds with return value 0  
Pressione qualquer tecla para continuar. . .
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Perceba que a aplicação só parou de executar quando o número oculto foi encontrado, este tipo de estrutura de repetição não entra em um *loop* infinito, pois, em algum momento, o número oculto será descoberto, parando o processo é apresentado na tela qual é o número, conforme demonstrado na imagem acima.

3.3 Estrutura do *do/while*

Vimos como é útil utilizar uma estrutura de repetição; uma derivação da *while* é a estrutura *DO --- WHILE* que também é considerada como um laço de repetição, ambas têm a sua composição bem parecida, entretanto, com alguns pequenos detalhes.

A estrutura **do while** é uma estrutura do tipo laço condicional, isto é, o loop baseia-se na análise de uma condição. Essa estrutura é utilizada quando temos um número indefinido de repetições e precisamos que o teste condicional seja realizado após a execução do trecho de código. Nesse tipo de estrutura, o trecho de código é executado pelo menos uma vez, pois o teste condicional é realizado no fim. (ASCENCIO e CAMPOS, 2010 p. 67)

Os pontos de maior diferença entre as duas estruturas estão no fato de entrada e teste condicional, na estrutura *while*, o laço só é executado se a expressão condicional inicial for atendida, caso contrário, este nem é executado, podendo assim nem ser executado no programa. A estrutura *do -- while* já não testa a

condição de entrada, desta forma, seu laço será executado ao menos uma vez, seu teste condicional não é de entrada, mas, sim, de saída do laço.

Veja a sintaxe do laço de repetição:

Figura 64. Título Composição textual do laço *Do ... While*

```
do{  
    <bloco de código a ser executado no loop>  
}while(condição);
```

Fonte: Os autores (2019).

Para lhe ajudar a compreender o funcionamento desta estrutura vejamos o código-fonte em linguagem C, em que o usuário digita a quantidade de repetição desejada com a estrutura *do -- while* para que execute e informe por linha cada laço. Observamos o código a seguir:

Figura 65. Algoritmo de exemplo DO --- WHILE

```
#include <stdio.h>  
#include <stdlib.h>  
#include <locale.h>  
  
main(void){  
    setlocale(LC_ALL, "Portuguese");  
    int numero, i=0;  
    printf("Digite a quantidade de repetições: ");  
    scanf("%d", &numero);  
    do{  
        i++;  
        printf("%dº Número\n", i);  
    }while (i < numero);  
    printf("\nFim da Estrutura DO -- WHILE\n\n");  
}
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Note que após o usuário digitar um determinado número, a estrutura *do -- while* já é executada, não sendo validada a entrada, o laço ficará efetuando as

repetições até que a condição $i < \text{numero}$ seja atendida. Vejamos o algoritmo em execução.

Figura 66. Resultado do programa de exemplo DO --- WHILE

```
Digite a quantidade de repetições:7
1º Número
2º Número
3º Número
4º Número
5º Número
6º Número
7º Número

Fim da Estrutura DO -- WHILE

-----
Process exited after 2.441 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Refleta

Quando não informamos uma variável que permita controlar o laço de repetição, podemos criar uma situação inesperada em nosso programa. O que acontece quando o laço de repetição não encontra uma condição de parada?

Esta estrutura de repetição é interessante de se utilizar quando pretendemos desenvolver um código que force o laço, uma utilização seria o menu de um programa, pois este sempre tem que se repetir a cada iteração e ao ser executado, já deve ficar disponível para o usuário.

Indicação De Recurso Didático

Algoritmos: Lógica Para Desenvolvimento de Programação de Computadores
Jayr Figueiredo de Oliveira e José Augusto N. G. Manzano



(Autores)

Descrição: Este livro abrange os principais conceitos de programação de computadores, incluindo a norma ISO 5807:1985 (E) e importantes fundamentos, como entrada, processamento, saída, tipos de dados, variáveis, constantes, operadores aritméticos e expressões aritméticas. Explica tomada de decisão, laços condicional e incondicional, programação com matrizes, técnicas de ordenação e busca, uso de registros e uma maneira de incorporar - em uma única matriz - dados de tipos diferentes. A organização de um programa em sub-rotinas complementa o ensino, abordando procedimentos, funções e passagens de parâmetro. A obra apresenta, ainda, medidas de complexidade, fundamentos de otimalidade e backtracking, bem como ações de busca de padrões em strings.

4. MÚLTIPLAS CONDIÇÕES E OPERADORES LÓGICOS

Conforme visto no tópico anterior, a estrutura de repetição *while* testa a expressão antes de executar o seu *loop*, vimos uma outra estrutura que trabalha de forma bem parecida, as estruturas de desvio condicional IF e IF - ELSE, entretanto, uma estrutura *while* se utiliza apenas de operadores relacionais não lógicos, não sendo possível efetuar a validação de duas expressões para forçar o laço de repetição.

Na estrutura *while*, podemos implementar condições para que a estrutura do *loop* seja acionada e que o *loop* continue em execução, até que esta condição

seja satisfeita, para isso, podemos utilizar como forma de controle operadores lógicos e relacionais, associados ao `if` que interferem diretamente no *loop*, veja o exemplo desenvolvido em linguagem C a seguir.

Figura 67. Algoritmo de seleção de participantes

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

int main(){
    setlocale(LC_ALL, "Portuguese");
    int H = 0, M = 0;
    int participantes = 0, idade;
    char sexo[2];

    printf("Para esta pesquisa você deve informar o sexo do participante H ou M e sua idade\n\n");

    while(M<3){
        printf("Qual o sexo do participante? ");
        scanf("%s", &sexo);

        printf("Qual a idade do participante? ");
        scanf("%d", &idade);

        //estrutura IF - ELSE
        if((strcmp(sexo,"M")==0 || strcmp(sexo,"m")==0) && idade>18){
            M++;
        }else{
            H++;
        }
        participantes++;
    }
    printf("Temos %d mulheres com idade acima de 18 anos, em um grupo de %d participantes.", M,
    participantes);
}
```

Fonte: Os autores (2019, programa utilizado DEV C++)

O algoritmo apresentado ficará em execução até que a quantidade informada de mulheres na pesquisa seja 3, mas para selecionar cada participante, este deve preencher alguns requisitos como sendo mulher e com idade acima de 18 anos, assim, é computada a variável que é o parâmetro *do while*. Vejamos agora o código fonte em execução:

Figura 68. Retorno do algoritmo de seleção de participantes

```
Para esta pesquisa você deve informar o sexo do participante H ou M e sua idade
Qual o sexo do participante? H
Qual a idade do participante? 15
Qual o sexo do participante? H
Qual a idade do participante? 26
Qual o sexo do participante? M
Qual a idade do participante? 17
Qual o sexo do participante? M
Qual a idade do participante? 63
Qual o sexo do participante? M
Qual a idade do participante? 25
Qual o sexo do participante? H
Qual a idade do participante? 25
Qual o sexo do participante? H
Qual a idade do participante? 45
Qual o sexo do participante? M
Qual a idade do participante? 27
Temos 3 mulheres com idade acima de 18 anos, em um grupo de 8 participantes.
-----
Process exited after 29.07 seconds with return value 76
Pressione qualquer tecla para continuar...
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Notamos que a quantidade de participantes informados foi acima de 3, pois totalizou 8 integrantes, entretanto, apenas 3 deles se encaixavam nos pré-requisitos, gerando dados para validação da expressão da estrutura de repetição *while*, mesmo não conseguindo efetuar uma condição múltipla no *while*, podemos nos utilizar de outras estruturas para deixar o algoritmo mais interessante e completo.

4.1 Exemplo: Repetição Controlada por Contador

Podemos controlar as iterações de um laço de repetição por meio de variáveis de controle, sendo que, a cada iteração, este é incrementado para validar a condição a ser testada em cada laço de repetição. Para que isso aconteça, inicializamos a variável de controle com um determinado valor para iniciar o laço de repetição e a saída dele dependerá exclusivamente da condição associada a esta variável, vejamos a seguir a composição do *while* com esta composição:

Figura 69. Composição da estrutura de repetição While com variável de controle

```
declaração da variável de controle

while (condição){
    <bloco de código a ser executado>

    função de incremento
}
```

Fonte: Os autores (2019).

Você pode perceber, aluno, que a estrutura ficou praticamente igual à do *while* convencional, adicionando apenas os elementos para os contadores de controle.

Para compreendermos como podemos utilizar esta estrutura em linguagem C, podemos desenvolver um algoritmo que efetue o laço por 20 vezes, apresentando apenas os números que sejam pares em cada linha, vejamos o código fonte a seguir:

Figura 70. Algoritmo *while* utilizando o contador para controle

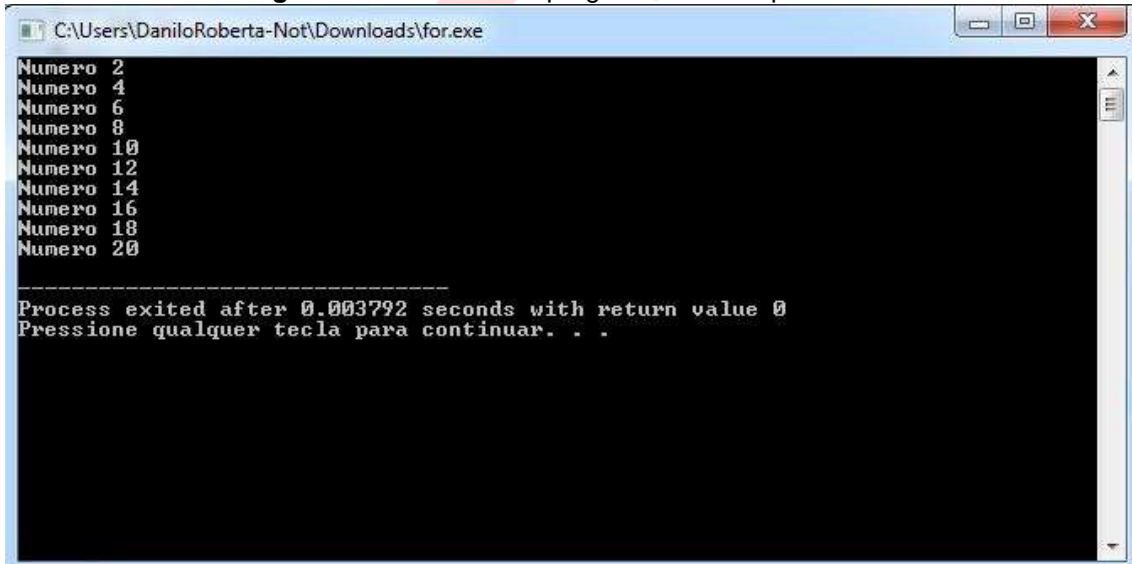
```
#include <stdio.h>
int i;
main(){
    i=1; //inicialização da variável de comparação
    while(i<=20){ //condição
        if(i % 2 == 0) printf("Numero %d \n", i);
        i++; // incremento
    }
}
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Ao analisarmos o código-fonte apresentado, percebemos que o laço de repetição efetua a validação do *loop* com a expressão $i \leq 20$, isso quer dizer que a estrutura de repetição só é executada caso esta condição for verdadeira e a estrutura fica em *loop* até que esta expressão não seja atendida, a cada nova iteração a variável de controle é incrementada $i++$, isso irá acontecer por 20 vezes até que a variável de controle tenha o valor de 20.

Neste código-fonte temos também a estrutura condicional IF, na qual é validade a expressão $i \% 2 == 0$, esta condição faz com que apenas os números que sejam par entrem em desvio condicional e sejam impressos na tela. Vejamos a seguir o código-fonte em execução.

Figura 71. Resultado do programa de exemplo WHILE



```
C:\Users\DaniloRoberta-Not\Downloads\for.exe
Numero 2
Numero 4
Numero 6
Numero 8
Numero 10
Numero 12
Numero 14
Numero 16
Numero 18
Numero 20
-----
Process exited after 0.003792 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Fonte: Os autores (2019, programa utilizado DEV C++)

A utilização de controladores incrementais para efetuar os laços de repetição é feita quando temos como padrão a quantidade de repetições necessárias, na imagem apresentada como sendo o resultado do algoritmo, podemos notar que foi impresso na tela apenas 10 linhas, mas a quantidade exata de repetições foi 20, conforme definido anteriormente, para este tipo de caso, a variável de controle foi excelente, não indicamos o uso de controladores se não souber a quantidade exata de repetições.

4.2 Exemplo: Repetição Controlada por Flags e Valores de Entrada

Na estrutura *do -- while*, podemos associar a expressão condicional ao valor de entrada, um elemento digitado pelo usuário, assim como apresentado na estrutura *while*, o que muda é o ponto de saída na estrutura *while*, vimos como exemplo um algoritmo no qual o usuário tinha que descobrir um número oculto, vejamos este mesmo exemplo, mas com a estrutura *do -- while*.

Figura 72. Algoritmo de exemplo *DO --- WHILE*

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

int main(){
    setlocale(LC_ALL, "Portuguese");
    int nOculto, nUsuario = 0;
    nOculto = 67;

    printf("Descubra qual o número oculto para sair do programa.");
    printf("\nVocê não tem um limite de chances, o número está entre 0 e 100.\n\n");

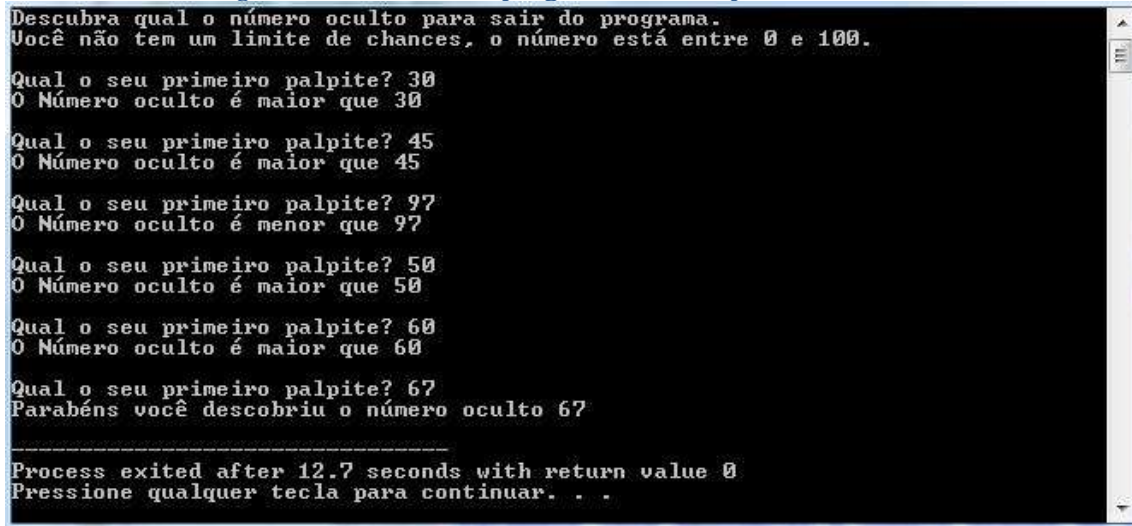
    do{
        printf("Qual o seu primeiro palpite? ");
        scanf("%d", &nUsuario);
        // estrutura IF - ELSE
        if(nUsuario == nOculto){
            printf("Parabéns você descobriu o número oculto %d \n", nOculto);
            break;
        }else{
            if(nOculto > nUsuario){
                printf("O número oculto é maior que %d \n\n", nUsuario);
            }else{
                printf("O número oculto é menor que %d \n\n", nUsuario);
            }
        }
    }while(nUsuario != nOculto);
}
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Neste algoritmo, quando for executado, o usuário só irá conseguir sair do *loop* até digitar o número correto, a cada repetição, será apresentada uma dica se o

número é maior ou menor do que o digitado. Perceba que a estrutura está praticamente igual à do exemplo *do while*, a diferença é que não precisamos iniciar ou testar as variáveis no início do loop. Vejamos a seguir o algoritmo em execução:

Figura 73. Resultado do programa de exemplo *DO --- WHILE*



```
Descubra qual o número oculto para sair do programa.  
Você não tem um limite de chances, o número está entre 0 e 100.  
  
Qual o seu primeiro palpite? 30  
O Número oculto é maior que 30  
  
Qual o seu primeiro palpite? 45  
O Número oculto é maior que 45  
  
Qual o seu primeiro palpite? 97  
O Número oculto é menor que 97  
  
Qual o seu primeiro palpite? 50  
O Número oculto é maior que 50  
  
Qual o seu primeiro palpite? 60  
O Número oculto é maior que 60  
  
Qual o seu primeiro palpite? 67  
Parabéns você descobriu o número oculto 67  
  
-----  
Process exited after 12.7 seconds with return value 0  
Pressione qualquer tecla para continuar. . .
```

Fonte: Os autores (2019, programa utilizado DEV C++)

Visualmente, o usuário não percebe que está em um laço de repetição até que a pergunta do palpite se repita sucessivamente. A cada nova iteração é requisitado um palpite e informado se o número é maior ou menor do que o digitado.

Enquanto o usuário não digitar o número correto este irá se manter no *loop*, a verificação da expressão condicional acontece após o usuário digitar algum número, caso este seja o número correto, o algoritmo para a sua execução e apresenta para o usuário a mensagem de que ele acertou o número.

Indicação De Recurso Didático
Algoritmos. Técnicas de Programação



Autores: José Augusto N. G. Manzano, Ecivaldo Matos, André Evandro Lourenço

Edição: 2. ed. – São Paulo: Érica, 2015.

Trecho para leitura:

Resumo: Voltado a estudantes de cursos técnicos e profissionalizantes, o livro abrange conceitos de programação de computadores, incluindo a norma ISO 5807:1985 (E) e conceitos importantes, como entrada, processamento, saída, tipos de dados, variáveis, constantes, operadores aritméticos e expressões aritméticas. Explica tomada de decisão, laços condicional e incondicional, programação com matrizes, técnicas de ordenação e busca, uso de registros e uma maneira de incorporar – em uma única matriz – dados de tipos diferentes. A organização de um programa em sub-rotinas complementa o ensino, abordando procedimentos, funções e passagens de parâmetro.

Disponível em: Biblioteca Virtual da Unifamma

CONCLUSÃO

Vimos nesta unidade os conceitos sobre estruturas de repetição. As estruturas de repetição são utilizadas para a repetição de trechos de códigos específicos, podemos observar que ao utilizar um laço de repetição não há necessidade da criação de diversas variáveis e rotinas.

As estruturas de repetição estudadas têm alguns pontos de diferença, como a forma de execução de seu laço de repetição. A estrutura *While* efetua uma validação da expressão antes de entrar no laço, a *Do -- While* valida apenas a expressão para a saída do laço, consequentemente, o laço é repetido ao menos uma vez, já a estrutura *for* executa um número determinado e incrementação. Devemos atentar se sobre todas as estruturas é utilizada alguma variável de controle, sem estas variáveis, podemos gerar o que chamamos de *loop* infinito, travando e inviabilizando o programa. De fato, não existe uma estrutura melhor do que a outra, mas sim aquela que melhor atende às necessidades do código.

Como resposta, a nossa pergunta norteadora, compreendemos que ao se utilizar de uma estrutura de repetição, podemos deixar o código-fonte final menor por conta dos laços de repetição, como consequência também podemos tornar o nosso programa otimizado.

Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**. 5. ed. Pearson Prentice Hall. São Paulo 2010.

DEITEL, H. M.; DEITEL, P. J. **C: como programar**. 6 ed. São Paulo, Pearson Prentice Hall. 2011.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. Pearson Pretice Hall. São Paulo, 2008.

SCHILDT, Herbert. **C completo e total**. 3. ed. rev. e atual. São Paulo: Makron Books, 1997.

