Há repetição em todos lugares, e nada é encontrado apenas uma vez no mundo.

Johann Wolfgang von Goethe

1 O mundo dá voltas - e algumas vezes ao contrário

Em vários momentos, na programação, se torna necessário repetir um trecho de um programa um determinado número de vezes. Nesse caso, pode ser criado um laço de repetição que efetue o processamento de um determinado trecho, tantas vezes quantas forem necessárias. Na estrutura de repetição *while* o teste é efetuado antes da execução dos comandos, agora veremos a estrutura de repetição com teste no final.

Esta estrutura de repetição é em tudo idêntica à a estrutura de repetição com teste no início (comando *while*). A diferença é que o teste é feito após o processamento da ação. O teste (da expressão lógica) sucede a ação (teste no final; depois de executar o bloco de comandos).Em C++, uma estrutura de repetição deste tipo é a instrução *do..while* (figura 1).

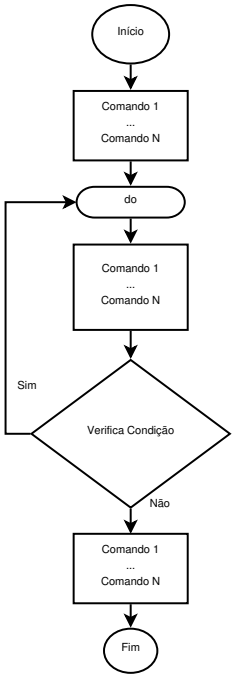

Figura 1: Fluxograma de Repetição com teste no final.

1.1 (Re)Introduzindo o *play again* no seu jogo

Como em todo jogo, é interessante dar a opção para o jogador tentar novamente a sua sorte. Então, para um jogo excitante (ao menos o nosso exemplo irá mostrar isto) é necessário perguntar ao jogador se ele deseja jogar novamente. Para tanto, ele irá informar ”S” ou ”N”. O programa 1 implementa o *Play Again*:

Programa 1: Possibilitando ao jogador tentar novamente.

```
1  // Introduzindo o play again
   // programa_001.cpp
   #include "biblaureano.h"

   int main()
6  {
     char again;

     do
     {
11     cout << "\n**Este jogo é muito excitante**" << endl;
       again = readChar("Você quer jogar novamente ? (s/n): ");
     } while( again == 's' );

     cout << "Game over!" << endl;
16   return 0;
   }
```

A figura 2 apresenta o fluxograma do programa 1.

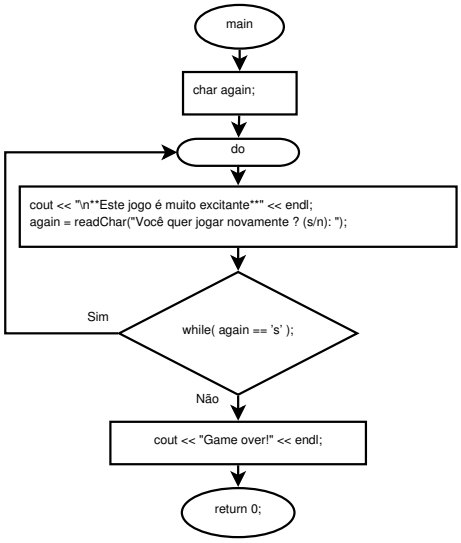

Figura 2: Fluxograma do programa 1.

1.2 Entendendo o programa

No início do programa, foi declarado uma variável para controlar a repetição.

```
  char again;
```

Na sequência, foi utilizada a estrutura *do..while* (faça-enquanto) para verificar se a condição é verdadeira.

```
do
{
 ......
} while( again == 's' );
```

Como a variável *again* contém o valor "s", a sequência de comandos dentro do { e } (que delimita o bloco de comandos do *do..while*) é executado.

O comando *do...while* diferencia-se do comando *while* somente em um detalhe. O bloco de comando indicado é sempre executado pelo menos uma vez. Após a execução do bloco a condição é testada. Caso seja verdadeira, o bloco continua a ser executado. A execução passará para o próximo comando somente quando a condição retornar falso.

```
    cout << "\n**Este jogo é muito excitante**" << endl;

    again = readChar("Voce quer jogar novamente ? (s/n): ");
```

Repare no que final, é solicitado ao jogador se ele deseja continuar o jogo, bastando informar "s" ou "n". Neste momento, chega-se ao final do bloco de comando (delimitado pelo } - fecha chaves) e comando *while* é avaliado e verificado a condição, sendo verdadeiro, o processo ocorre novamente (controle volta para o início da estrutura *do*).

2 Utilizando o *do..while* para validar entrada de dados

Em vários exemplos e exercícios vistos até o momento, muitos dos dados de entrada (informados pelo usuário) não foram validados. Utilizando a estrutura *do..while* é possível validar as informações do usuário, e no caso das informações fornecidas não forem válidas, solicitar que o usuário informe novamente. Veja o programa 2.

Programa 2: Validando as informações do usuário.

```
// Introduzindo o play again
// mas validando a entrada
// programa_002.cpp
#include "biblaureano.h"

int main()
{
  char again;
  do
  {
    cout << "\n**Este jogo é muito excitante**" << endl;

    do
    {
      again = readChar("Você quer jogar novamente ? (s/n): ");
    } while( again != 's' && again != 'n');
```

```
17
      cout << "\n************************" << endl;
    } while( again == 's');
    cout << "Game over!" << endl;
    return 0;
22  }
```

A figura 3 apresenta o fluxograma do programa 2.

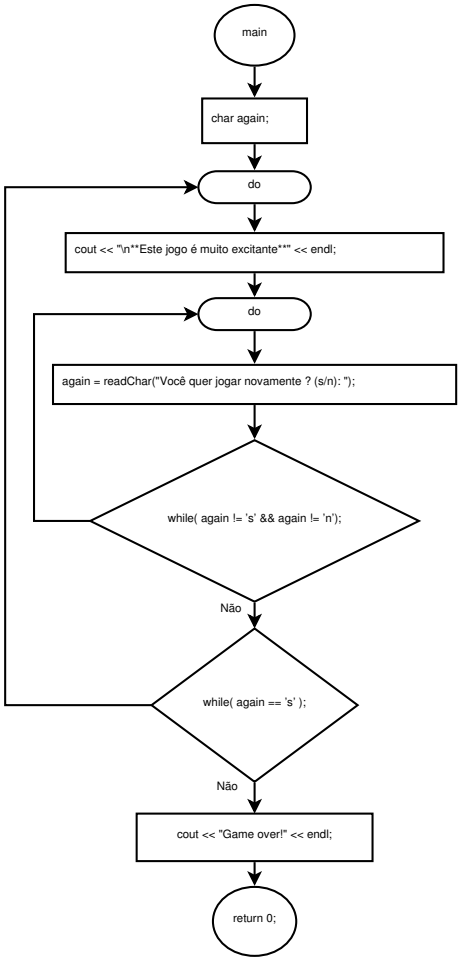

Figura 3: Fluxograma do programa 2.

2.1 Analisando o programa

Repare que o programa somente continuará a sua execução, se e somente o usuário entrar com um valor válido (neste caso 's' ou 'n').

```
    do
```

```
    {
      again = readChar("Você quer jogar novamente ? (s/n): ");
    } while( Again != 's' && Again != 'n');
```

E somente após o valor ter sido validado, é que o restante do programa é executado. Neste caso, é impresso uma linha e verificado se o programa deve continuar em execução:

```
      cout << "\n************************" << endl;
  } while( again == 's');
```

3 Exercícios

1. Faça um programa que mostre a tabuada de 2 (2x1=2, 2x2=4,...).
2. Calcule e mostre a soma de todos os números entre 1 e 100.
3. Calcule e mostre a soma dos números pares entre 1 e 100.
4. Calcule e mostre a soma dos números impares entre 1 e 100.
5. Apresente o quadrado de cada um dos números pares entre 1 e 1000.
6. Apresente todos os números divisíveis por 5 que sejam maiores do que 0 e menores ou iguais a 200.
7. Escreva um programa que gere os números de 1000 a 1999 e escreva aqueles que dividido por 11 dão resto igual a 5.
8. Faça um programa que leia 2 números inteiros e positivos. O programa deverá mostrar a multiplicação de todos os números entre o intervalo (inclusive). Exemplo: se for digitado 15 e 19 deverá ser apresentado o resultado da multiplicação de 15 * 16 * 17 * 18 * 19.
9. Escreva um algoritmo que leia 10 valores quaisquer. A seguir, mostre quantos deles estão dentro do intervalo [10,20] e quantos estão fora do intervalo, mostrando essas informações.
10. Faça um programa que verifique se um número é primo. Obs: É considerado número primo somente os números que são divisíveis por 1 e por ele mesmo (exemplos: 1, 2, 3, 5, 7,11,13,17,19,23,29....). Observação 2: Todos os números primos são ímpares (exceto o 2).
11. Fazer um programa que calcule e imprima o fatorial de um número fornecido pelo usuário. Repetir a execução do programa tantas vezes quantas o usuário quiser. Lembre-se que o resultado do cálculo de um fatorial pode ser um número *grande* (Exemplo: Fatorial de 8 = 40320). Para quem não lembra fatorial, fatorial de $5! = 5 * 4 * 3 * 2 * 1$.
12. Ler o nome de um aluno e suas duas notas A e B, e após calcular a média ponderada entre estas notas (A tem peso 1 e B tem peso 2). Repetir este procedimento para uma turma composta por cinco alunos.
13. Escreva um programa para que calculem o resultado de cada uma das seguintes séries com 50 termos:
    - 1 + 3 + 9 + 27 + 81 + ...
    - 1/1 + 2/4 + 3/9 + 4/16 + 5/25 + ...
    - 1 - 2 + 3 - 4 + 5 - 6 +.....
14. Faça um programa que receba 30 números e mostre quantos positivos, quantos negativos, quantos pares, quan-

tos ímpares e qual é o maior e o menor número informado nesta lista.

15. Elaborar um programa que leia dois números inteiros quaisquer, em linhas diferentes, e que determine (mostre) se são AMIGOS ou NÃO AMIGOS. O programa deve também, para cada número, mostrar seus divisores em linhas diferentes, e a soma dos mesmos. Números amigos e não amigos são aqueles que a soma dos seus divisores resulta no outro número. Exemplo:220 e 284 Os divisores de 220 são:1+2+4+5+10+11+20+22+44+55+110=284 Os divisores de 284 são:1+2+4+71+142=220