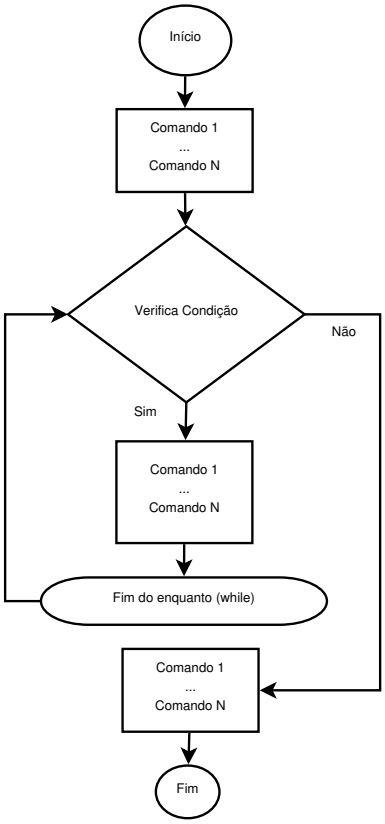
> Há repetição em todos lugares, e nada é encontrado apenas uma vez no mundo.
>
> Johann Wolfgang von Goethe

1 O mundo dá voltas - emaranhando o mundo

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. Os laços de repetição também são conhecidos por *loopings*.

Neste caso, também há a necessidade de tomar uma decisão com base no valor lógico de uma expressão. No entanto, a mesma ação será executada repetidamente enquanto o resultado da expressão lógica se mantiver verdadeiro. O teste (da expressão lógica) precede a ação (teste no início; antes de começar o bloco de comandos). O teste é importante porque funciona como uma *condição de parada* (quando a condição for falso) dos laços de repetições (figura 1). Em C++, uma estrutura de repetição deste tipo é a instrução *while*.

Figura 1: Fluxograma de Repetição com teste no início.



A instrução *while* indica o início de um *loop* e recebe como parâmetro uma condição. Essa condição recebe o nome especial de condição de parada, pois quando essa condição falhar (devolver *false*), o *loop* é encerrado.

```
1  while( condição de parada )
   {
     instrução 1;
     instrução 2;
     ....
6    instrução n;
   }
```

Se traduzirmos para o português o termo *while* como *enquanto*, fica fácil entender a instrução: enquanto a condição de parada for verdadeira, execute instrução 1, instrução 2,..., instrução n.

1.1 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.

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

   int main()
   {
     char again = 's';
8
     while( again == 's' )
     {
       cout << "\n**Este jogo é muito excitante**" << endl;

13     again = readChar("Você quer jogar novamente ? (s/n): ");
     }
     cout << "Game over!" << endl;

     return 0;
18 }
```

1.2 Entendendo o programa

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

```
  char again = 's';
```

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

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

Teste lógico: usamos testes lógicos para saber quando parar a repetição, ou seja, estabelecemos uma condição para a repetição continuar, caso contrário o programa ficaria preso na repetição eternamente. Neste caso verificamos o conteúdo da variável *again* até que o seu conteúdo seja diferente de 's'.

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

```
    cout << "\n**Este jogo e 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 o controle volta automaticamente para o comando *while* (onde será novamente verificado a condição, sendo verdadeiro, o processo ocorre novamente). A figura 2 apresenta o fluxograma do programa 1.

2 Formas de Controle das Estruturas de Repetição

As estruturas de repetição possuem um forma diferenciada para controlar a quantidade de repetições a serem executadas. Estes controles podem ser genericamente classificados em automáticos ou controlados pelo usuário do programa.

Automático : Uma variável auxiliar contará quantas vezes será executado o conjunto de comandos (bloco de repetição), sem interferência direta do usuário. O bloco será repetido sempre a quantidade de vezes prevista pelo desenvolvedor do programa. Em outras palavras, o programador define quantas vezes aquele bloco será repetido.

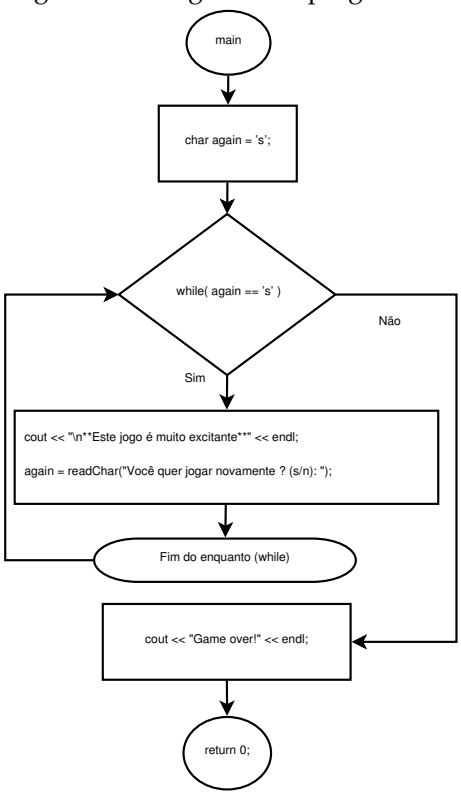
Controlado pelo usuário: O programa sempre respeitará a solicitação do usuário, executando um bloco de repetição quantas vezes forem solicitadas pelo usuário, respeitando a lógica existente no programa desenvolvido. Em outras palavras, o usuário controlará quantas vezes um bloco será executado.

Existem 2 tipos de repetições:

Definidas e garantidas: ou seja, um mesmo trecho de programa será executado um número pré-determinado de vezes.

Indefinida e sem garantia: ou seja, um trecho de programa poderá ser executado indefinidamente, não havendo garantia do término da execução do programa.

Figura 2: Fluxograma do programa 1.



Normalmente, todo programa com controle *automático* tem sua estrutura de repetição *definida e garantida*. Por sua vez, todo programa *controlado pelo usuário* tem sua estrutura de repetição *indefinida e sem garantia*.

O programa 1 é um exemplo de programa cujo controle *pertence ao usuário* e logo, sua estrutura é *indefinida e sem garantia*. O controle pertence ao jogador, pois somente *ele* decide quando parar de jogar, e, como o controle pertence ao usuário, o programa não tem como saber *quantas* vezes o jogador irá repetir a execução do programa.

Mas o que é um programa com *controle automático* e com *repetição definida e garantida*? Para isto, temos que entender o que são variáveis *acumuladoras* e *contadoras*.

2.1 Variáveis Contadoras, Somadoras ou Acumuladoras

Contagens ou somas e multiplicações acumulativas são procedimentos a que se recorre com frequência nas mais diversas situações e seu uso é comum em códigos de programas.

Uma contagem é usada para, por exemplo, contabilizar o número de execuções de um laço de repetição já concluídos, determinar o número de vezes que um determinado valor ocorre em uma determinada sequencia de valores ou monitorar o número de caracteres já consumidos de um fluxo de caracteres. Somas ou produtos acumulativos são frequentes em cálculos de somatórias, produtórias ou consolidações de resultados (totais, médias, ...).

Contadores e somadores usualmente têm atribuído o valor 0 (zero) como valor inicial (elemento neutro da soma) e, sempre que apropriado, são acrescidos por mais um termo (o valor 1 (um) no caso de contadores ou outro valor

quando se trata de somadores/acumuladores).

Variáveis utilizadas para o cálculo de produtórias são inicializadas usualmente com o valor 1 (um) (elemento neutro da multiplicação/divisão) e, sempre quando necessário, são atualizadas com o resultado do seu valor corrente multiplicado por um novo termo.

Portando, uma variável do tipo *acumulador* é qualquer variável que recebe um valor inicial constante (geralmente 0) e recebe ela mesma mais uma outra variável ou qualquer valor (pode não ser fixo) em algum ponto do programa.

Variável do tipo acumuladora

```
int totalMontrosMortos = 0;
totalMonstrosMortos = totalMonstrosMortos + totalMonstrosAliensAzuis;
```

Logo, uma variável do tipo *contador* é uma variável que recebe um valor inicial e depois é incrementada por um *valor constante*, ou seja, recebe o mesmo valor.

Variável do tipo contadora

```
int contador = 0;
contador ++;
```

Considere o seguinte problema:

Alterar o programa 1 para que conte quantas vezes o usuário repetiu o jogo.

O objetivo do problema é ilustrar o uso de uma variável usada para fins de *contagem*. Logo, uma variável desta natureza é, usualmente, chamada de *contador*. Os motivos para contar algo são os mais diversos:

- Contar quantas vezes um determinado laço de repetição foi executado;
- Contar quantos números pertencentes a um determinado intervalo de valores foram lidos;
- Contar o número de termos já somados no cálculo de uma somatória;
- ....

No problema proposto pede-se que sejam contados a quantidade de vezes que o jogador repetiu o jogo. A estratégia a ser adotada é declarar uma variável com um valor inicial 0 (zero) e sempre que o conteúdo do jogo for apresentado ao jogador, somar 1 (um). No final do jogo, informar quantas vezes ele jogou o jogo (programa 2).

Programa 2: Contando quantas vezes o jogador repetiu a jogada.

```
// Introduzindo o play again
// contando as coisas
// programa_002.cpp
#include "biblaureano.h"

int main()
{
  int qtdJogadas = 0;
  char again = 's';

  while( again == 's' )
  {
    cout << "\n**Este jogo é muito excitante**" << endl;
```

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

    qtdJogadas++; //QtdJogadas = QtdJogadas + 1;
  }

  cout << "Você jogou " << qtdJogadas << " vezes!" << endl;
  cout << "Game over!" << endl;

  return 0;
}
```

Analisando o programa 2, percebe-se que o controle das vezes que o jogador repetiu a jogada é:

```
qtdJogadas++;
```

Lembre-se que variáveis contadoras devem começar com um valor inicial definido, normalmente 0 (zero) e o seu incremento/decremento é constante.

2.1.1 Voltando a estrutura com repetição automática e com fim definido e garantido

Após ter entendido o conceito de variável *contadora* fica fácil lançar um problema a ser resolvido.

Criar um jogo, onde seja solicitado ao usuário número e de acordo com a resposta, seja mostrada uma mensagem.

Vamos alterar o nosso programa da *Fortuna perdida!* e incluir a possibilidade de jogar 3 vezes o mesmo jogo, apenas solicitado os dados iniciais novamente (programa 3).

Programa 3: Jogo *Fortuna perdida!* em nova versão.

```
// Uma nova versão da fortuna perdida
// programa_003.cpp
#include "biblaureano.h"

int main()
{
  const int PECAS_OURO = 900;
  int aventureiros, mortos, sobreviventes;
  string lider;

  int qtdJogadas = 0;

  while( qtdJogadas < 3 )
```

```
14    {
15
16      //pegando as informações
17      cout << "Bem vindo ao 'Fortuna Perdido!'.\n\n";
18      aventureiros = readInt("Por favor entre com a quantidade de aventureiros:");
19
20      // cálculo do número de mortos
21      mortos = randomico(1,aventureiros-1);
22
23      sobreviventes = aventureiros - mortos;
24
25      lider = readString("Entre com o seu último nome:");
26
27      //contando a história
28      cout << "\n\tUm grupo de " << aventureiros << " bravos aventureiros ";
29      cout << "recebeu uma missão - encontrar e recuperar um tesouro guardado ";
30      cout << "pelo homem do saco. ";
31      cout << "Este grupo era liderado pelo famoso " << lider;
32      cout << ". Um grande mestre das armas.";
33
34      cout << "\n\tAo longo do caminho, vários sacis atacaram o grupo. ";
35      cout << "Os aventureiros, seguindo os comandos do seu bravo líder " << lider;
36      cout << ", derrotaram vários dos atacantes. Mas houve um custo. ";
37      cout << "Dos " << aventureiros << " aventureiros, " << mortos;
38      cout << " deram suas vidas pela missão. ";
39      cout << "Os " << sobreviventes << " restantes continuaram fortes e unidos ";
40      cout << " para completar a missão.";
41
42      cout << "\n\tO grupo estava cheio de esperança e cansados, afinal, ";
43      cout << "o ouro prometido compensaria todo o esforço. ";
44      cout << "Após uma última e árdua luta, o homem do saco foi derrotado. ";
45      cout << "Os aventureiros acharam as " << PECAS_OURO << " peças de ouro.";
46      cout << "O líder " << lider << " deu a cada aventureiro ";
47      cout << (PECAS_OURO / sobreviventes) << " peças de ouro.";
48      cout << "As peças restantes (" << (PECAS_OURO%sobreviventes) << ") ele guardou ";
49      cout << " para si (junto com as que já ganhou na divisão é claro).";
50
51      cout << "\n\tDepois de cumprirem a brava missão, o grupo seguiu seu caminho em paz.";
52
53      cout << endl << "-------------------------------------" << endl;
54      qtdJogadas++;
55    }
56
57    cout << "Você jogou " << qtdJogadas << " vezes!" << endl;
58    cout << "Game over!" << endl;
59
60    return 0;
61  }
```

Observe o valor inicial da variável *qtdJogadas* e a condição para a parada:

```
1   int qtdJogadas = 0;
2
3   while( qtdJogadas < 3 )
```

```
{
    ....
}
```

Isto significa que a variável conterá os valores 0,1 e 2 (o que daria 3 tentativas). Para iniciar a variável com o valor 1 (um) seria necessário modificar a condição de parada:

```
int qtdJogadas = 0;

while( qtdJogadas <= 3 ) //poderia ser qtdJogadas < 4
{
    ....
}
```

Neste caso, a variável conterá os valores 1,2 e 3.

Muito cuidado no valor inicial de uma variável contadora e a condição utilizada.

2.2 Combinando repetições com decisões

Naturalmente é possível combinar estruturas de repetição com decisões. Você pode utilizar estruturas de repetição com decisões dentro ou estruturas de decisão com repetições dentro. Você pode utilizar estruturas de repetição dentro de outras estruturas de repetição. O programa 4 exemplifica a utilização de estruturas de decisão dentro de uma estrutura de repetição, a figura 3 apresenta graficamente o programa 4.

Programa 4: Tentando acertar um número.

```
// programa_004.cpp
#include "biblaureano.h"

int main()
{
  int qtdJogadas = 0;
  char again = 's';

  while( again == 's' )
  {
    int numeroJogador, numeroSorteado;
    numeroJogador = readInt("Entre com o seu palpite:");
    numeroSorteado = randomico(1,10);

    if(numeroSorteado == numeroJogador)
    {
       cout << "Parabéns, você acertou o palpite!" << endl;
    }
    else
```

```
    {
       cout << "Você erro, o número sorteado foi "
           << numeroSorteado << endl;
    }

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

    qtdJogadas++;
  }

  cout << "Você jogou " << qtdJogadas << " vezes!" << endl;
  cout << "Game over!" << endl;

  return 0;
}
```

2.3 Crítica - garantindo que os dados digitados são válidos

Usamos uma crítica para conferir se foi digitado um valor correto, usando para isso um teste lógico, dentro de uma repetição (laço), de onde só sairá quando for digitado um valor desejado, ou seja, obrigando que seja digitado um valor válido.

Vejamos um exemplo onde é necessário calcular a a pontuação média de um jogador (não existe placares negativos). Vamos supor que cada jogador têm três placares diferentes e temos cinco jogadores. Para cada jogador é necessário verificar se os seus placares estão corretos. A figura 4 demonstra o fluxograma do programa 5:

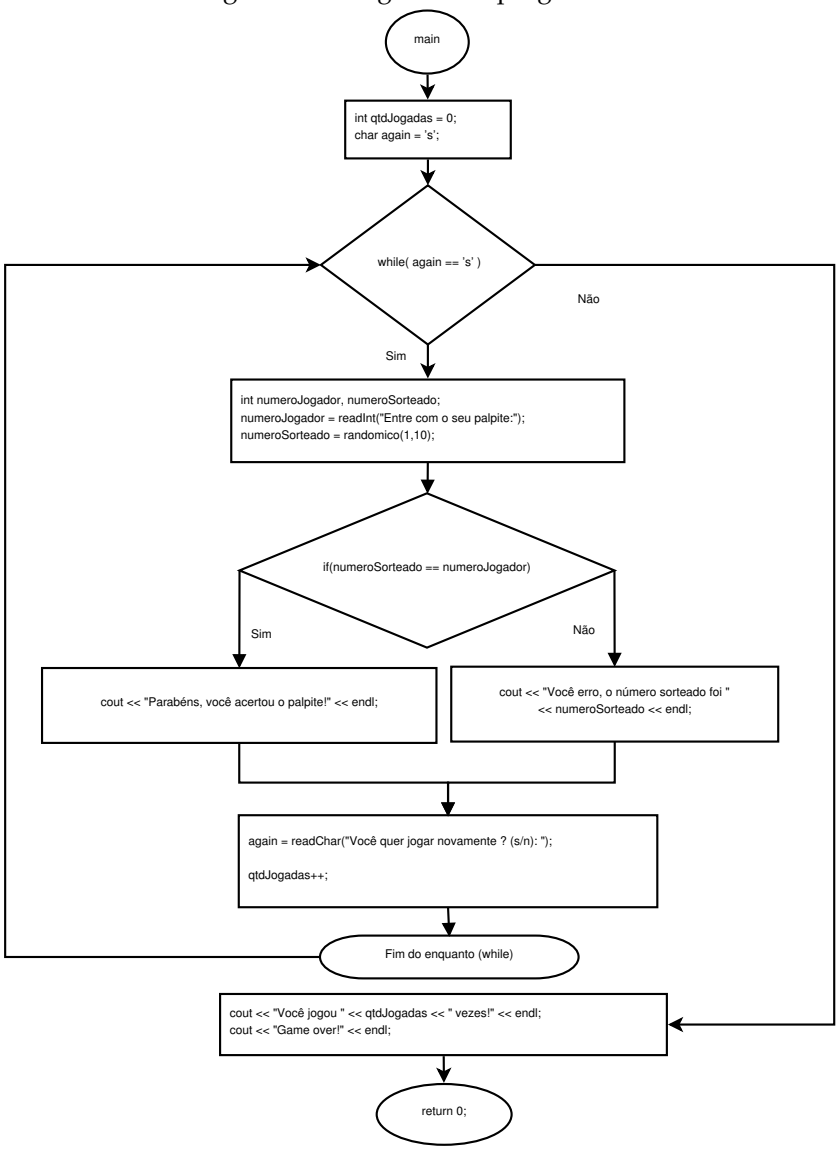
Programa 5: Verificando se os dados digitados são válidos.

```
// programa_005.cpp
#include "biblaureano.h"

int main()
{
  int jogadores = 0;
  while( jogadores < 5)
  {
    //lê os placares
    int placar1 = readInt("Entre com 1 placar:");
    int placar2 = readInt("Entre com 2 placar:");
    int placar3 = readInt("Entre com 3 placar:");

    //verifica se algum placar é negativo
    while( placar1 < 0 || placar2 < 0 || placar3 < 0)
    {
       cout << "Um ou mais placares estão com o valor inválido!!!!"<<endl;
       placar1 = readInt("Entre com 1 placar:");
       placar2 = readInt("Entre com 2 placar:");
       placar3 = readInt("Entre com 3 placar:");
    }
    //calcula e mostra a média
    float media = (placar1+placar2+placar3)/3.0;
    cout << "Média deste jogador:" << media << endl;
```

Figura 3: Fluxograma do programa 4.

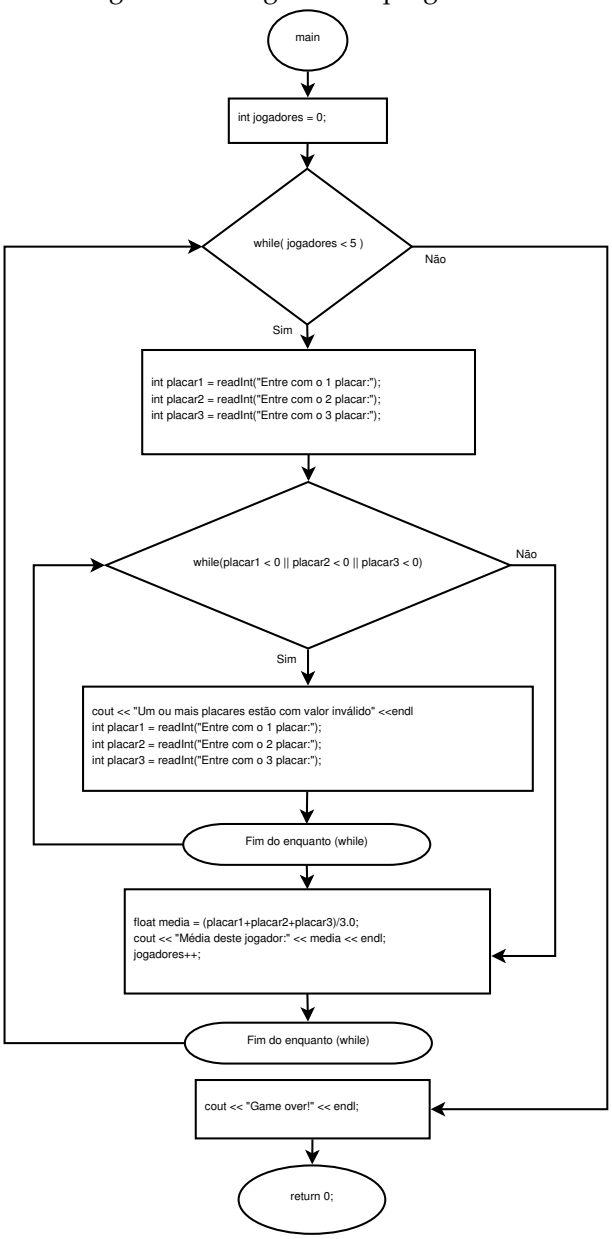



```
    jogadores++; //incrementa a quantidade de jogadores
26  }

    cout << "Game over!" << endl;

    return 0;
31 }
```

Figura 4: Fluxograma do programa 5.



3 Determinando o maior e/ou menor valor numa lista de valores

Em muitos programas surge a necessidade de determinarmos qual o maior ou o menor valor dentro de um conjunto de valores e, para isto, não existe uma estrutura especial, apenas utilizamos os conhecimentos que já aprendemos.

Escreva um programa para ler a nota dos alunos e escrever a nota mais alta, ou seja, a maior nota entre as 10 notas lidas.

Quando sabe-se os limites dos valores possíveis, ou seja, por exemplo com as notas sabemos que os valores serão de 0 a 10, então sabe-se quais são os valores limites (o valor mínimo e o valor máximo), não teremos nota menor que 0 e nem nota maior que 10. Nesses casos é mais fácil descobrir o maior ou o menor valor, pois pode-se inicializar a variável *maior*, por exemplo, com o valor 0 e a variável *menor* com o valor 10 que funcionará perfeitamente.

```
int nota,maior,menor;
maior = 0,
menor = 10;

int contador = 1
while(contador<=10)
{
  nota = readInt("Entre com a nota:");
  if( nota > maior )
  {
    maior = nota;
  }
  if( nota < menor )
  {
     menor = nota;
  }
  ++contador;
 }
```

Acontece que se não sabe-se os valores dos limites aí complica um pouco, pois não sabemos com que valor vamos inicializar as variáveis para comparação.

Leia um conjunto de dez valores e determine qual o maior e menor valor lido.

Então temos que inicializar tanto a variável *maior* quanto a *menor* com o *primeiro valor lido* e depois vamos comparando os próximos valores lidos com o primeiro!

```
int maior, menor, valor;
valor = readInt("Entre com um valor:");
menor = valor;
maior = valor;

int contador = 2; //o primeiro termo já foi lido
while(contador<=10)
{
  valor = readInt("Entre com outro valor:");
  if( valor > maior )
  {
    maior = valor;
  }
  if( valor < menor )
  {
     menor = valor;
  }
  ++contador;
}
```

Os nomes *maior* e *menor*, são apenas exemplos, pode-se denominar as variáveis que serão usadas para os testes como quiser.

4 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, quantos 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
16. Escrever um programa que lê um valor N inteiro e positivo e que calcula e escreve o valor de E.

    $$E = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{N!}$$
17. Construir um programa que calcule a média aritmética de vários valores inteiros positivos, lidos externamente. O final da leitura acontecerá quando for lido um valor negativo.
18. Escreva um programa que calcule a média dos números digitados pelo usuário, se eles forem pares. Termine a leitura se o usuário digitar zero (0).
19. Escrever um programa que leia um número n que indica quantos valores devem ser lidos a seguir. Para cada número lido, mostre uma tabela contendo o valor lido e o fatorial deste valor.
20. Escrever um programa que lê um número não determinado de pares de valores **m**,**n**, todos inteiros e positivos, um par de cada vez, e calcula e escreve a soma dos **n** inteiros consecutivos a partir de **m** inclusive.
21. Escrever um programa que lê 5 pares de valores **a**, **b**, todos inteiros e positivos, um par de cada vez, e com $a \leq b$, escreve os inteiros pares de **a** até **b**, incluindo o a e o b se forem pares.
22. Mostrar o quadrado de todos os números inteiros de 1 a 20.