Decisões são mais fáceis quando não há mais escolhas.

Narasimha Rao

1 A verdade encadeada

Existem casos em que é necessário se estabelecerem verificações de condições sucessivas. Quando uma ação é executada, ela poderá ainda estabelecer novas condições, isso significa condições dentro de condições. Esse tipo de estrutura poderá ter diversos níveis de condição, sendo chamados de aninhamentos ou encadeamentos.

O segundo encadeamento pode ser tanto para uma condição verdadeira quanto uma condição falsa.

Você está escrevendo um jogo online. Neste jogo, você quer mostrar uma mensagem personalizada para os usuários (desde que o usuário e a senha estejam corretas). Considere a tabela 1:

Tabela 1: Usuários e suas senhas.

Usuário	Senha	Mensagem a ser mostrada
Marcos	pacato123	E ai Marcos ? Tudo em riba ?
José	mariobros	Fala José! Quais as novidades ?
Maria	gasolina	Oi Maria!

Logo, percebe-se que para mostrar a mensagem, é necessário que o usuário e a senha estejam corretas. Logo, são duas condições a serem avaliadas e ambas devem ser verdadeiras. Uma forma de fazer isto seria assim:

```
1  if( usuario == "Marcos")
   {
     if( senha == "pacato123" )
     {
       cout << "E ai Marcos ? Tudo em Riba ?" << endl;
6    }
   }
```

Mas esta não é a solução adequada. Para resolver este problemas, a linguagem C++ nos fornece os *operadores lógicos*.

2 Operadores lógicos e a tabela verdade

Um valor lógico pode estar em apenas um dos estados: verdadeiro (*true*) ou falso (*false*) e os seus valores estão presentes em qualquer comparação (lembre-se dos operadores relacionais).

Logo, os *operadores lógicos* consideram dois operandos como valores lógicos (verdadeiro e falso) e realizam a operação binária correspondente. Estas operações quando aplicadas irão retornar *true* ou *false*, conforme o resultado seja verdadeiro ou falso. Na tabela 2 é apresentados os operadores lógicos da linguagem C++.

Tabela 2: Usuários e suas senhas.

Operador	Equivalência	Significado	Exemplo
&&	E lógico (ou conjunção)	Retorna verdadeiro caso ambas as partes sejam verdadeiras.	usuario == "Marcos"&& senha == "pacato123"
||	OU lógico (ou disjunção)	Basta que uma parte seja verdadeira para retornar verdadeiro.	usuario == "Marcos"|| usuario == "Maria"
!	Não lógico (ou negação)	Inverte o estado, de verdadeiro passa para falso e vice-versa.	!(usuario == "Marcos")

Considerando uma proposição qualquer como sendo uma variável lógica, uma relação ou uma expressão lógica composta. Duas proposições podem ser combinadas por um operador lógico e formar uma nova proposição.

Considere as seguintes variáveis e proposições:

- $A \leftarrow 10$ (variável);
- $B \leftarrow 5$ (variável);
- $C \leftarrow 27$ (variável);
- $D \leftarrow -5$ (variável);
- $A > B$ (proposição);
- $C < D$ (proposição);

No caso, se utilizarmos o operador $\&\&$ (e lógico), estamos fazendo uma *conjunção* das proposições originais. A conjunção de duas proposições é verdadeira se e somente **ambas** as proposições forem verdadeiras.

Como exemplo, temos a proposição composta $A > B \;\&\&\; C < D$. O resultado desta proposição é falso (*false*), embora $10 > 5$ seja verdadeiro, a proposição $27 < -5$ é falsa e a regra da conjunção diz que para uma proposição composta seja considerada verdadeira **ambas** deverão ser verdadeiras..

Se utilizarmos o operador || (ou lógico), estamos fazendo uma *disjunção* das proposições originais. A disjunção de duas proposições é verdadeira se e somente se, pelo menos, **uma** delas for verdadeira. Utilizando o mesmo exemplo, temos a a proposição composta $A > B \;||\; C < D$. O resultado desta proposição é verdadeira (*true*), pois mesmo que a proposição $27 < -5$ seja falsa, a regra da disjunção diz que para uma proposição composta seja considerada verdadeira **apenas uma** das proposições precisa ser verdadeira.

Caso seja utilizando o operador ! em uma proposição, estamos fazendo uma *negação* da proposição original. Então, se negarmos a proposição $C < D$ ($!(27 < -5)$) obteremos o valor verdadeiro (*true*).

Como a função de um operador lógico é gerar um único resultado (verdadeiro ou falso), chegamos a *tabela verdade*

(tabela 3).

Tabela 3: Tabela verdade.

A	**B**	**A && B**	**A || B**	**! A**
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

2.1 Trabalhando com o operador lógico &&

Voltando ao problema da mensagem personalizada, a tabela 4 apresenta algumas possíveis combinações de uso do operador &&.

Tabela 4: Possíveis combinações do operador &&.

usuario == "Marcos"	**senha == "pacato123"**	**usuario == "Marcos"&& senha == "pacato123"**
true	true	true
true	false	false
false	true	false
false	false	false

O programa 1 faz uso deste operador. Neste programa também está sendo utilizado o operador de negação (!) e o operador de ou lógico (||).

Programa 1: Trabalhando com os operadores lógicos.

```
// Um jogo em rede
// Mostrando as mensagens personalizadas
// programa_001.cpp
#include "biblaureano.h"

int main()
{
  string usuario, senha;

  //vamos controlar também o nível de acesso
  enum ACESSO { SEM_ACESSO, CONVIDADO, ESPECIAL};
  ACESSO nivelAcesso = SEM_ACESSO;

  usuario = readString("Informe o seu usuario:");
  senha = readString( usuario + ", entre com sua senha:");

  if( usuario == "Marcos" && senha == "pacato123" )
  {
    nivelAcesso = ESPECIAL;
```

```
    cout << "E ai Marcos ? Tudo em riba ?" << endl;
  }

  if( usuario == "José" && senha == "mariobros" )
  {
    nivelAcesso = ESPECIAL;
    cout << "Fala José! Quais as novidades?" << endl;
  }

  if( usuario == "Maria" && senha == "gasolina" )
  {
    nivelAcesso = ESPECIAL;
    cout << "Oi Maria!" << endl;
  }

  // ou lógico
  if( usuario == "convidado" || senha == "convidado")
  {
    nivelAcesso = CONVIDADO;
    cout << "Bem vindo usuário sem registro!" << endl;
  }

  //se NivelAcesso continuar com o valor 0 (falso)
  if(!nivelAcesso) //lembre-se, negar um valor significa inverter o valor
  {
    cout << "Você não está autorizado a jogar!!!" << endl;
  }

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

2.2 Trabalhando com o operador lógico ||

Como visto no programa 1, o trecho de código:

```
  if( usuario == "convidado" || senha == "convidado")
  {
    nivelAcesso = CONVIDADO;
    cout << "Bem vindo usuário sem registro!" << endl;
  }
```

Verifica se o usuário ou a senha são de usuários convidados. Neste caso, se um dos valores informado for a palavra *convidado*, o usuário poderá jogar, porém com um acesso menos privilegiado. A tabela 5 apresenta algumas possíveis combinações de uso do operador ||.

2.3 Trabalhando com a negação

Como visto no programa 1, os trechos de códigos:

Tabela 5: Possíveis combinações do operador ||.

usuario == "convidado"	senha == "convidado"	usuario == "convidado"|| senha == "convidado"
true	true	true
true	false	true
false	true	true
false	false	false

```
ACESSO nivelAcesso = SEM_ACESSO;
```

```
if(!nivelAcesso)
{
  cout << "Você não está autorizado a jogar!!!" << endl;
}
```

Cria a variável *nivelAcesso* com o valor zero (*SEM_ACESSO*). Logo depois, verifica se o usuário pode continuar jogando.

Um valor zero em C++ representa o falso e um valor diferente de zero representa verdadeiro. Logo, ao negar o valor zero (falso), obtém-se verdadeiro.

2.4 As prioridades da verdade

Assim como os operadores aritméticos, a linguagem C++ estabelece uma prioridade de avaliação para os operadores lógicos. O operador mais prioritário é a negação (!). O operador de conjunção (&&) tem maior prioridade que o operador de disjunção (||).

Utilize parênteses para facilitar o entendimento do seu código. O trecho de código abaixo:

```
if( usuario == "Maria" && senha == "gasolina" )
{
  nivelAcesso = ESPECIAL;
  cout << "Oi Maria!" << endl;
}
```

Poderia ser escrito assim:

```
if( (usuario == "Maria") && (senha == "gasolina") )
{
  nivelAcesso = ESPECIAL;
  cout << "Oi Maria!" << endl;
}
```

3 Os diversos caminhos da verdade

Caso seja necessário realizar operações baseadas em um valor de uma expressão ou variável em vez de se construir para isto uma cadeia de *if...else...if...else..if...else* pode-se utilizar o comando de seleção múltipla *switch...case*:

```
switch (expressão)
{
  case constante 1:
  {
    bloco de comandos 1...;
    break;
  }
  case constante 2:
  {
    bloco de comandos 2...;
    break;
  }
  .......
  default:
  {
    bloco de comandos n...
  }
}
```

Exemplos da vida real:

- Caso ***TEMPO*** seja igual a:
  - **Chuva**: Então fico em casa;
  - **Sol**: Vou à praia;
  - **Encoberto**: Vou ao cinema.
- Caso ***HORÁRIO*** seja igual a:
  - **08 horas**: Vou trabalhar;
  - **12 horas**: Vou almoçar;
  - **18 horas**: Vou para casa;
  - **20 horas**: Vou jantar;
  - **Qualquer outro horário:** Vou dormir.

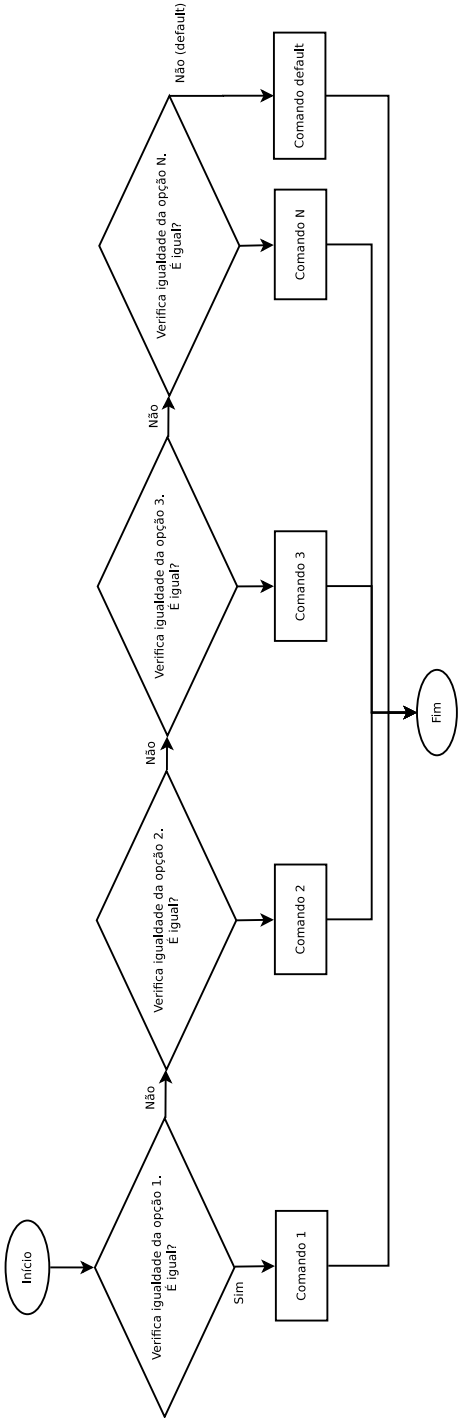
A estrutura de decisão *switch* é utilizada para testar, na condição, uma única expressão, que produz um resultado, ou, então, o valor de uma variável, em que está armazenado um determinado conteúdo. Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula.

Neste caso, a decisão não é feita com base numa expressão lógica porque há mais do que 2 resultados possíveis. Também só são processadas a ação ou as ações encontradas numa via (figura 1). Exista a opção de se utilizar a cláusula *default*, para os casos em que nenhum *case* anterior tenha sido selecionado. A cláusula *default* é de uso opcional.

Inicialmente o valor da expressão é avaliado. Depois é feita uma comparação com cada valor colocado na lista de opções. Caso o valor seja coincidente o bloco ligado a opção será executada. Na prática, o uso do *switch*, é um típico exemplo de estrutura de *estrutura de seleção com **n** alternativas*.

O uso do comando *switch* só é válido para testes de igualdade, não sendo possível realizar uma comparação relacional, por exemplo.

Figura 1: Fluxograma com Decisão com N alternativas.



O programa 2 apresenta o funcionamento da estrutura *switch*.

Programa 2: Escolhendo um nível de dificuldade.

```
// Os vários caminhos da verdade
// uso do switch
// programa_002.cpp
#include "biblaureano.h"

int main()
{
  int dificuldade;

  cout << "Escolha o nível de dificuldade:" << endl;
  cout << "\t1 - Fácil\n\t2 - Médio\n\t3 - Normal\n\t4 - Difícil" << endl;
  dificuldade = readInt();

  // criando os níveis de dificuldade, começando em 1
  enum DIFICULDADE {FACIL=1, MEDIO, NORMAL, DIFICIL};

  switch(dificuldade)
  {
    case FACIL:
    {
      cout << "Nível Fácil" << endl;
      break;
    }
    case MEDIO:
    {
      cout << "Nível Médio" << endl;
      break;
    }
    case NORMAL:
    {
      cout << "Nível Normal" << endl;
      break;
    }
    case DIFICIL:
    {
      cout << "Nível Difícil" << endl;
      break;
    }
    default:
    {
      cout << "Opção não existe!" << endl;
    }
  }

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

Inicialmente o valor da expressão é avaliado. Depois é feita uma comparação com cada valor colocado na seção *case*. Caso o valor seja coincidente o bloco ligado ao *case* será executado. Convém ressaltar que a execução continuará na

ordem que os comandos aparecem, indiferentemente se eles fazem parte de outro *case*. Para interromper a execução deve-se utilizar a cláusula *break*, indicando que deve ser interrompido a execução e passar a executar os comandos após o *switch*.

Existe a possibilidade de colocar uma condição para que, se nenhum *case* foi selecionado, um bloco seja executado. A palavra *default* indicará este bloco padrão a ser executado. A figura 3 demonstra, em fluxograma, o que ocorre no programa 2.

Figura 2: Fluxograma do programa 2.

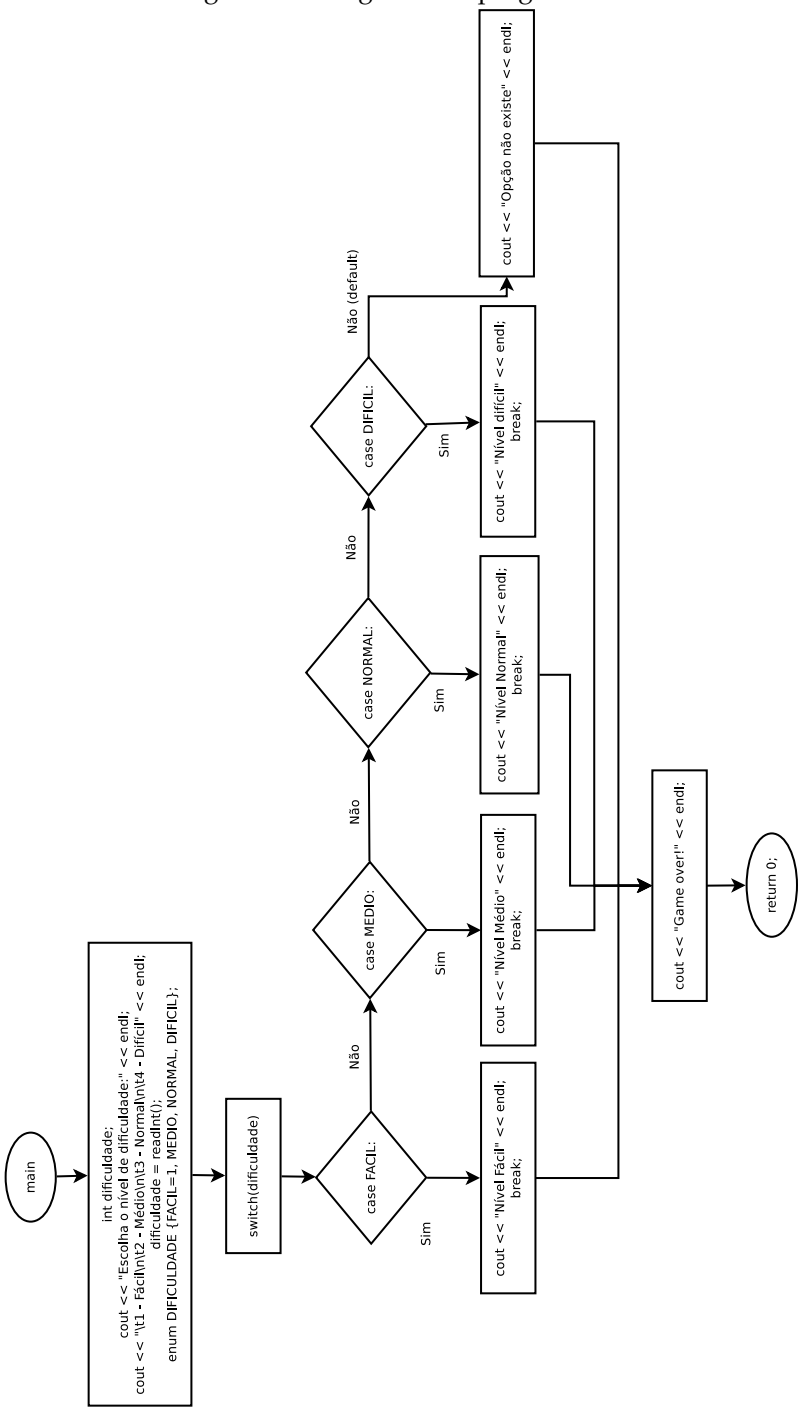


Figura 3: Fluxograma do programa 2.

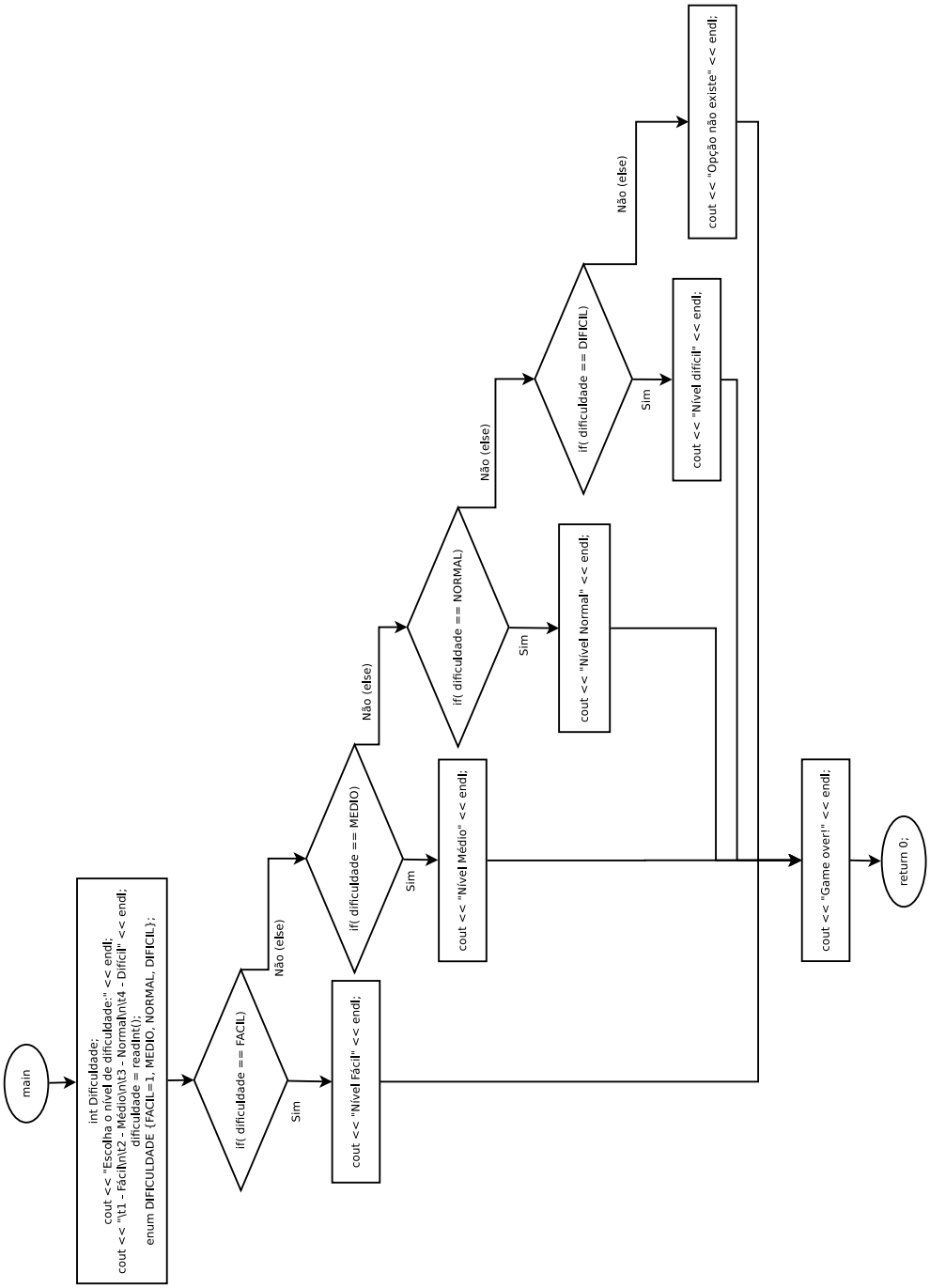
No programa 3 vemos uma versão do programa 2, mas utilizando uma cadeia de *if..else*. Compare a figura 3 com a figura 4.

Programa 3: Escolhendo um nível de dificuldade (versão com *if..else*).

```
// Os vários caminhos da verdade
// a outra alternativa de uso
// programa_003.cpp
#include "biblaureano.h"

int main()
{
  int dificuldade;
  cout << "Escolha o nível de dificuldade:" << endl;
  cout << "\t1 - Fácil\n\t2 - Médio\n\t3 - Normal\n\t4 - Difícil" << endl;
  dificuldade = readInt();

  // criando os níveis de dificuldade, começando em 1
  enum DIFICULDADE {FACIL=1, MEDIO, NORMAL, DIFICIL};

  if(dificuldade == FACIL)
  {
    cout << "Nível Fácil" << endl;
  }
  else
  {
    if(dificuldade == MEDIO)
    {
      cout << "Nível Médio" << endl;
    }
    else
    {
      if(dificuldade == NORMAL)
      {
        cout << "Nível Normal" << endl;
      }
      else
      {
        if(dificuldade == DIFICIL )
        {
          cout << "Nível Difícil" << endl;
        }
        else
        {
          cout << "Opção não existente!" << endl;
        }
      }
    }
  }

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

No comando *switch* somente podem ser utilizados valores inteiros ou que possam ser convertidos para inteiros, como uma variável do tipo *char*.

Figura 4: Fluxograma do programa 3.



4 Exercícios

1. Resolva a tabela verdade abaixo:

Tabela 6: Exercício de tabela verdade.

A	B	C	A && (B || C)	A && ! C	B || C || A	A && B && C	C && (B || A)
true	false	true					
false	true	true					
true	true	false					
true	false	false					

2. Tendo como dados de entrada o sexo do jogador (M - masculino e F - feminino), construa um programa que *corrija* a pontuação, aplicando uma bônus. Utilize as seguintes fórmulas:
   - Para homens: bônus = pontos * 0.05
   - Para mulheres: bônus = pontos * 0.07
3. Seu jogo é orientado para jogadores com mais de 16 anos. Pergunte a idade do jogador e verifique se ele pode jogá-lo.
4. No seu jogo, é necessário ocorrer a troca de moedas entre os jogadores. Para tal, é necessário uma conversão, conforme abaixo:
   - 1 moeda de ouro = 50 moedas de prata;
   - 1 moeda de prata = 50 moedas de cobre;

   Pergunte ao jogador quantas moedas (quantidade e tipo) ele tem e o tipo da moeda que ele quer receber. Por exemplo, se ele informar que tem 50 moedas de cobre, ele deve receber 1 moeda de prata. Se ele tiver 1 moeda de ouro, ele pode receber 50 moedas de prata ou 2500 moedas de cobre.
5. Tendo como dados de entrada a altura e o sexo de uma pessoa (M - masculino e F - feminino), construa um algoritmo que calcule seu peso ideal, utilizando as seguintes fórmulas:
   - Para homens: (72.7 * altura) - 58
   - Para mulheres: (62.1 * altura) - 44.7
6. Ler um número e verificar se ele é múltiplo de 5.
7. Obtenha a idade de uma pessoa e escreva se ele é maior ou menor de idade.
8. Leia dois números e informe se eles são iguais ou diferentes.
9. Verifique se um número par ou ímpar.
10. Ler dois valores e verificar a divisão do maior pelo menor.
11. Escreva um programa que lê um valor em reais e um tipo de moeda (1 - dólar ou 2 - euro) e faz a conversão do valor para a moeda solicitada.
12. Ler um número e informar se ele é positivo ou negativo.
13. Elabore um algoritmo que dada à idade de um nadador classifique-o em uma das seguintes categorias (utilize o comando *if..else*):

- Infantil A = 5 a 7 anos
- Infantil B = 8 a 11 anos
- Juvenil A = 12 a 13 anos
- Juvenil B = 14 a 17 anos
- Adultos = Maiores de 18 anos

14. Elabore um algoritmo que dada à idade de um nadador classifique-o em uma das seguintes categorias (utilize o comando *switch*):
    - Infantil A = 5 a 7 anos
    - Infantil B = 8 a 11 anos
    - Juvenil A = 12 a 13 anos
    - Juvenil B = 14 a 17 anos
    - Adultos = Maiores de 18 anos
15. Faça um programa que leia a receita e a despesa de uma firma e que imprima lucro ou prejuízo.
16. Uma empresa está contratando para seu quadro de funcionários pessoas com as seguintes características: Maior de 18, universitário e com conhecimentos de inglês. Construa um algoritmo que leia os dados e informe se o candidato está apto ou não ao cargo.
17. Um imposto é calculado com base na seguinte tabela:

Até 1.200,00	isento	
de	1.201,00 a 5.000,00	10%
de	5.001,00 a 10.000,00	15%
acima de	10.000,00	20%.

    Implemente um programa que calcule os impostos a pagar para um valor de acordo com a faixa. Peça para o usuário informar este valor.
18. Implemente um programa que receber 3 valores numéricos, X, Y e Z, e verificar se esses valores podem corresponder aos lados de um triângulo. Em caso afirmativo, informar ao usuário se o triângulo é equilátero, isósceles ou escaleno.
19. Faça um programa que lê do usuário um caractere e informa se ele é uma vogal, uma consoante ou não é uma letra.