**CICLO DE VIDA**

Conjunto de etapas de trabalho pela quais uma MDS organiza suas atividades, de forma sequencial ou paralela

Uma MDS, genericamente possui o seguinte clico de vida

**- LEVANTAMENTO DE DADOS**

**- ANÁLISE**

**- PROJETO**

**- IMPLEMTENAÇÃO**

**- TESTES**

De acordo com a isso 12207:1998 o ciclo de vida é a estrutura contendo processos atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso”

As etapas do ciclo de vida de um software são organizacionais e sequenciam de forma cronológica e ordenada os passos de criação e desenvolvimento de aplicações.

**CICLO DE VIDA – ETAPAS**

**DEFINIÇÃO -** Deve-se conhecer aa situação atual e fazer a identificação do problema para buscar uma resolução dele. É na definição que você fará a modelagem dos processos e análise do sistema. O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software.

**DESENVOLVIMENTO** - Esta etapa envolve atividades relacionadas a design, prototipagem, codificação, testes, entre outras atividades que forem necessárias como por exemplo a integração com um outro sistema. É importante ressaltar que essas atividades devem seguir o que foi descrito nas etapas anteriores, pois é aí que entras as regras de negócio.

**OPERAÇÃO** - Nesta etapa o software já estará em produção e você dará o devido suporte aos usuários e corrigir possíveis bugs. Também entra a continuidade

**METODOLOGIAS TRADICIONAIS X METODOLOGIAS AGEIS NO GERENCIMANETO DE PROJETOS**

Atualmente, existem diferentes formas de gerenciar projetos, sejam eles de TI ou de qualquer outro segmento, mas o conceito de Métodos Ágeis é relativamente novo.  
Nos últimos anos, tem-se abordado muito a utilização desse modelo para demanda de gestão, principalmente na área de tecnologia.

Enquanto os métodos tracionais como não poderiam deixar de ser são mais conhecidos e gozam de uma cultura organizacional mais presente os métodos ágeis vem conquistando cada vez mais espaço

Os ***METODOS TRADICIONAIS*** se caracterizam pela burocracia e necessidade de realização de cada processo definido pela norma, ou pelo conjunto de boas práticas estabelecido no método. Conhecido também como método iterativo, o foco está nos processos, no monitoramento do andamento do projeto através das entras, ou seja, a ideia é de que só existe progresso quando tudo e entregue, apenas com 00% do projeto conclui é que o cliente atestará a satisfação com o produto/serviço. É um método de estrutura mais rígido pouco flexível a mudanças e com grande exaltação da figura do gerenciam de projetos. Nessa categoria se destacam as boas práticas oriundas do PMI ou CMMi.

Os METODOS AGEIS dão foco ao que se pode chamar de ambiente colaborativo pois tira a tenção do processo e contenta os esforços na contribuição do time, dos integrantes do projeto. É uma evolução dos métodos tradicionais.

É possível observar a diferença uma diferença radical em relação aos métodos antecessores na medida em que propõe a participação mais próxima do cliente, iterações mais curtas e testes automatizados. Porém é recomendando trabalhar com times menores dividir os projetos em equipes menores para otimizar o seu gerenciamento, facilitando a fluidez da informação

Os METODOS AGEIS se destacam por dar mais liberdade ao planejamento, estar abertos a mudanças positivas e proporcionar a gestão conjunta que concede autonomia de decisões alinhadas entre o time. Nessa categoria podem ser citados os métodos Extreme Programming também conhecidos como o XP e o Scum

**MANIFESTO AGIL**

Em 2001 um grupo de 17 renomados desenvolvedores de software agrupam e aprimoram os diversos conceitos de metodologias ágeis existentes e assinaram o “manifesto para o desenvolvimento ágil de software” que passou então a ser muito difundido pelas comunidades de desenvolvimento.

Esse manifesto basicamente reúne 4 valores e 12 princípios

Os ***valores*** determinam que se deve priorizar:

**1 – os indivíduos e as interações entre eles** mais que os processos e as ferramentas

**2 – O software funcionando** mais do que a uma documentação completa e abrangente.

**3 – A colaboração com e dos clientes** mais do que as negociações de contratos

**4 – Resposta a mudanças** mais do que seguir o plano inicial

Quanto aos **princípios** são eles:

**1** – A maior prioridade é satisfazer o cliente através da entrega adiantada e contínua de software de valor

**2** – Aceitar mudanças de requisitos mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.

**3** – Entregar software funcionando com frequência preferencialmente em semanas

**4** – Cooperação diária entre pessoas que entendem do ’negócio’ e desenvolvedores

**5** – Projetos surgem através de indivíduos motivados entre os quais existe relação de confiança

**6** – A maneira eficaz e eficiente de transmitir informações são através de conversas

**7** – Software funcionais são a principal medida de progresso do projeto

**8** – Processos ágeis promovem um ambiente sustentável. Os patrocinadores desenvolvedores e usuários, devem ser capazes de manter esses passos constantes

**9** – Continua atenção a excelência técnica e bom design, aumenta a agilidade

**10** – Simplicidade é essencial. Cultivar a arte de maximizar a quantidade de trabalho que não precisou ser feito

**11** – As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizadas

**12** – Em intervalos regulares, o time reflete em como se tornar mais efetivo, então, ajustam e otimizam seu comportamento de acordo.

***LOGICA DE PROGRAMAÇAO***

Conjunto de regras e conceitos que norteiam a criação de um código escrito para uma máquina. Isso faz com que os dados recebidos por ela possam ser convertidos em informações que serão compreendidas por pessoas ou outras máquinas seguindo alguma regra preestabelecida

A lógica de um sistema e estruturada por algoritmos que são sequencias de ações que são executadas sobre dados, a fim de transformá-los em informações que gerem valor essas regras são estabelecidas por Modelos de negócio QUE PODEM SER:

**Regras para cadastro em uma universidade**

**Etapas para a realização de uma transação bancária**

**Perguntas que levam o paciente a indicar os sintomas de uma doença**

Para definir um fluxo de informações de maneiro logica em algum algoritmo temos as seguintes ferramentas:

**ESTRUTURAS DE CONTROLE:** determinam se uma ação pode ou não ocorrer Por exemplo: um carro trafega por uma via e encontra um semáforo. Se o sinal estiver vermelho, ele deve parar. S estiver verde, ele pode seguir. Se estiver amarelo, deve reduzir a velocidade ou avançar, caso esteja próximo o suficiente do semáforo.

Para definir um fluxo de informação de maneiro logica em algum algoritmo, temos as seguintes ferramentas:

**LAÇOS DE REPETIÇÃO:**  determina uma condição para que uma ação seja repetida. Uma vez encontrada deve ser interrompida, exemplo: dirija até chegar à faculdade

**ARMAZENAMENTO DE DADOS:** bancos de dados, arquivos de texto, memórias de dispositivos eletrônicos.

Para definir um fluxo de informação de maneira lógica em um algoritmo, temos as seguintes ferramentas:

**TIPOS DE DADOS:** categorização das informações q indicam como deverão ser processados por exemplo como texto, números inteiros, decimais, data etc.

**VARIAVEIS:** são utilizadas para armazenar os valores de dados enquanto são trabalhados pelo algoritmo, “x=2”, “x=y”.

**ALGORITMOS:**

Existem várias maneiras de escrever um algoritmo, por exemplo:

**Pseudocódigo:** que é uma forma de representar um modelo de algoritmo usando a língua falado, que no nosso caso a língua portuguesa **Fluxograma:** que é uma forma expressada em figuras simbolizando o fluxo de dados e os comandos do algoritmo

**CASE SENSITIVE**

**Professora/ PROFESSORA/ ProFeSSOra/ professora**  
Preste atenção nas letras maiúsculas e minúsculas. Para linguagens de programação que:  
**SÃO CASE-SENSITIVE:** as quatro palavras são interpretadas diferentes; **NÃO SÃO CASE-SENSITIVE:** as quatro palavras são iguais  
 **C# é case-sensitive**

**VARIAVEIS**

As variáveis são utilizadas para armazenar informações na memória em tempo de execução da aplicação, isso significa que essas informações estarão disponíveis enquanto a aplicação estiver em execução e, mais precisamente, enquanto a classe ou método em que ela foi criada estiver em execução.  
No **C#** toda variável deve ter: **modificador de acesso, tipo de dados e nome.**

**REGRAS PARA NOMEAR VARIAVEIS**

A regra para nomear uma variável é que o nome dela sempre comece por uma **letra** ou \_. No meio do nome podem-se usar números, mas não se devem usar caracteres especiais e não pode ser uma palavra reservada ou comandos, no C# esses comandos ficam da cor azul.

No exemplo, Int é o tipo e número é o nome da variável.  
Toda variável deve ter o modificador de acesso. No exemplo não tem um modificador porque no C# quando não há um modificador em uma variável é atribuído a ele o modificador padrão private.

**Modificadores:** no caso de variáveis, os modificadores definem a visibilidade delas, se elas poderão ser acessadas por outras classes que não seja a sua própria, se serão acessadas somente por classes derivadas a classe que ela está e assim por diante.  
**Public**: o acesso não e restrito  
**Protected**: O acesso é limitado as classes ou tipos derivados da classe que a variável está  
**Internal**: O acesso é limitado ao conjunto de módulos (assembly) corrente.  
**Protected Internal**: O acesso é limitado ao conjunto corrente ou tipos derivados da classe recipiente.  
**Private**: O acesso é limitado a classe que a variável está.

**TIPOS DE DADOS C#**

**C#** é uma linguagem de programação **fortemente tipada**, isso significa que todas as variáveis e objetos devem ter um tipo declarado. O valor atribuído a uma variável deve estar de acordo com o seu tipo declarado.

**Byte -** 0 ...255  
**sbyte** - -128 ...127  
**short** - -32,768 ..31,767  
**unshort** – 0 ..65,535  
**int** - -2,147,438,648 ..2,147,483,647  
**uint** – 0 ..4,294,967,295  
**long** - -9,223,372,036,854,775,808..9,223,372,036,854,775,807  
**ulong** – 0 ..18,446,744,073,709,551,615  
**float** - -3,402823e38 ..3,402823e38  
**double** - -179769313486232e308 ..1,79769313486232e308  
**decimal** - -79228162514264337593543950335 ..73228162514264337593543950335  
**char** – U+0000 ..U-ffff

**CONSTANTES**

Em algumas situações trabalhamos com algum valor que **não precisa ser alterado.** Por exemplo uma medida que é padrão. Em casos como esses, utilizamos constantes, que tem características semelhantes a uma variável, mas que tem de receber um valor em sua declaração que não será alterado.

**OPERADORES C#**

**Operador de atribuição**O símbolo igual (=) representa a atribuição de um valor a uma variável, onde ela e o valor atribuído devem obrigatoriamente ser de tipos compatíveis, ou seja, uma variável do tipo int por exemplo não pode receber um valor do tipo **string.**

**OPERADORES C#**

**Operadores aritméticos**os operadores aritméticos são os memos usados em cálculos matemáticos

|  |  |
| --- | --- |
| **+** | **Adição** |
| **-** | **Subtração** |
| **\*** | **Multiplicação** |
| **/** | **Divisão** |
| **%** | **Módulos (resto da divisão)** |

**OPERADORES C#**

**Operadores incrementais e decrementais**Os operadores incrementais e decrementais tem a função de aumentar ou diminuir exatamente em **1** o valor de uma variável.  
  
**Incremental automático (++)  
Decremental automático(--)  
Incremental definido (+=)  
Decremental Definido (-=)**

**OPERADORES C#**

**Operadores relacionais**os operadores relacionais comparam dois valores e retornam um valor booleano (**true** ou **false**).

|  |  |
| --- | --- |
| **==** | **Igual a / comparação** |
| **!=** | **Diferente de** |
| **>** | **Maior que** |
| **<** | **Menor que** |
| **>=** | **Maior do que ou igual a** |
| **<=** | **Menor do que ou igual a** |

**OPERADORES C#**

**Operadores lógicos**Os operadores lógicos trabalham como operandos booleanos e seu resultado também será booleano (**true** ou **false**), são usados **somente** em expressões lógicos.

|  |  |
| --- | --- |
| **&&** | **And = E** |
| **||** | **Or = Ou** |
| **!** | **Not = Não** |

**| - pipe**

**Projeto** - arquivo inteiro

**Solução** – “pedaços”

------------------------------------------------------------------------------------------------------

**Font** – nome tamanho fonte

**Fore color** – cor da fonte

**Name** – nome

**Text** – nome que aparece

**MaxLenght** – tamanho máximo de caracteres

**Start position** – onde começa na tela de execução

**back color** – cor do fundo

**BRIEFING**

Conjunto de informações básicas, instruções, diretrizes etc., elaborado para a execução de um determinado trabalho ou missão militar.  
Reunião onde se definem com essas instruções, diretrizes etc.

O briefing e um conjunto de informações passadas em uma reunião para o desenvolvimento de um trabalho, sendo muito utilizadas em administração e na publicidade

É uma peça fundamental para a elaboração

O briefing e um conjunto de perguntas ou atividades que servem para determinar como se dará o projeto elaboração e execução de determinada coisa (Projeto).

**QUESTÕES: BRIEFING INICIAL (BÁSICO)**

* **PÚBLICO - ALVO**
* **OBJETIVO DO WEBSITE**
* **IMAGEM QUE O CLIENTE DESEJA PASSAR AOS VISITANTES**
* **ANÁLISE NA CONCORRÊNCIA (BENCHMARKING)**

**BRIEFING FINAL DETALHADO**

* **Estratégia**
* **Informações gerais**
* **Público-alvo**
* **Conteúdo**
* **Aparência design**
* **Outras informações pertinentes**

**Dados elementares**

Contatos, como telefone e-mail site são informações básicas que todo briefing deve coletar, além disso, contato um breve histórico de sua empresa para que a agência entender melhor em que o negócio se encontra sua missão visão e valores

**Escopo do projeto**

E importante delimitar logo de início se o projeto será algo novo, feito do zero ou se será redesenho de algo já existente. Pois em casos distintos e que demandam informações diferentes para sua execução, informar isso é delimitar o escopo do projeto.

**DEMANDA ESPECÍFICA**

Uma identidade visual? Um Site? Uma animação? Um vídeo?

O prestador de serviço precisa saber o que exatamente é a demanda do projeto, pois ela influencia em questões especificas no projeto, cada serviço é diferente. Por exemplo, se for uma identidade visual, irá incluir papelaria? Quais itens na papelaria? Informar e documentar isso no briefing é necessário para o projeto e uma questão de segurança e garantia para o cliente e agência.

**PÚBLICO-ALVO**

Ainda mais importante do que conhecer a empresa conhecer o público-alvo para o qual o website, marca e demais peças estão sendo produzidos, pois eles serão os usuários/ consumidores. Informar quem soa ajuda a definir uma persona, o que é importante para um direcionamento de projeto, esse tipo de informação deve constar no briefing.

**BENCHMARKING**

Um ponto muito importante é conhecer o mercado, seu segmento, seus concorrentes e observar o que eles têm feito de bom e ruim para usar como referência para futuras ações e além de conhecer a concorrência também tentar extrair o seu diferencial a partir delas

**EXPECTATIVAS**

E imprescindível saber o que o cliente e a agência esperam do projeto, e quais os resultados que almejam, pois tais informações fazem diferença durantes o desenvolvimento e ainda motivam a equipe a tentar superar.

**O QUE NÃO É ESPERADO**

Assim como saber as expectativas, é bom saber o que não e esperado de forma alguma no projeto, por exemplo, cores desejadas, formas, mensagens. Não tem nada pior do que um cliente decepcionado.

**COMUNICAÇÃO**

Dialogar e essencial, pois nem sempre todas as informações necessárias para o projeto estão contidas nas respostas do briefing. Tente expressar os pensamentos mesmo quando a agência não inserir um espaço livre. (cuidado com excessos).

**ESTRUTURA DE CONTROLE OU CONDICIONAL**

Aprender a criar estruturas de condição é fundamental para programar a lógica de negócio de um sistema. Essas estruturas permitem a execução de diferentes linhas de código (diferentes fluxos de execução) a partir da avaliação de uma condição. Na linguagem C# temos dois recursos com esse propósito: **if/else** e **switch/case**.

**ESTRUTURA DE CONTROLE OU CONDICIONAL – IF/ELSE**

A estrutura condicional **if/else** é um recurso que indica quais instruções o sistema deve processar de acordo com uma expressão booleana. Assim, o sistema testa se a uma condição é verdadeira e então executa comandos de acordo com esse resultado.

**ESTRUTURA DE CONTROLE OU CONDICIONAL**

Caso a expressão booleana seja verdadeira as instruções entre chaves presentes no código 1 serão executadas caso contrário serão executadas as instruções presentes no código 2.

As chaves, ou delimitadores de bloco, tem a função de agrupar um conjunto de instruções. O uso desses delimitadores é **opcional** caso haja apenas uma linha de código ainda assim seu uso e recomendando, pois, essa pátria facilita a leitura e manutenção do código.

**ESTRUTURA DE CONTROLE OU CONDICIONAL – ELSE IF**

Complementar ao if/else temos o operador **else if** que traz uma nova condição a ser testada no caso de falha no teste da condição anterior.  
testamos a expressão booleana 1. Caso ela seja atendida, o bloco 1 é executado. Caso não seja, testamos a expressão 2. Sendo atendida, o bloco de código 2 é executado. Não sendo atendido o programa executa o bloco de código 3. Podemos criar vários else if, o que nos possibilita atender a cenários com três ou mais condições a serem avaliadas.

**EXERCICIO ELSE/IF**

Elabore um programa que verifica se um aluno foi **aprovado**, se está em **recuperação** ou se foi **reprovado**.Para ser **aprovado,** o **aluno precisa ter média maior ou igual (>=) a 7.** Para a **recuperação**, ele precisa ter **média menos (<)** **que 7 e (&&) maior ou igual (>=) a 5.** Por fim, para ser **reprovado,** precisa ter **média menor (<) do que 5.**

|  |  |
| --- | --- |
| **VERDE** | **APROVADO** |
| **VERMELHO** | **REPROVADO** |
| **AMARELO** | **RECUPERAÇÃO** |

**EXERCICIO ELSE/IF**

Desenvolva um algoritmo que solicite a entrada da idade de um determinado usuário. Se for menor que 18 anos exiba na cor vermelha "Sem Permissão". Caso seja maior ou igual a 18 anos exiba na cor verde "Permissão Concedida".

**Exercícios de Lógica (Estrutura Condicional)**

1. Elaborar um sistema onde o usuário digita o número do mês e exibir o nome do mês correspondente.

2. Elaborar um sistema onde o usuário digita um número e o sistema informa se o número é par ou ímpar.

3. Elaborar um sistema onde o usuário informa o preço de custo de um produto e calcular o valor de venda, através do percentual informado pelo usuário.

4. Elaborar um sistema onde o usuário digita 04 notas e exibe:

Maior Nota,

Menor Nota e

Média.

5. Elaborar um sistema para calcular o aumento de salário de acordo com as regras:

Se o salário digitado for maior que 5000 dar um aumento de 10%,

se o salário for menor que 5000 e maior que 2000 aumento de 20%

senão aumento de 30%.

vezes a % dividido por 100

***GIT E GITHUB CONFIGURANDO***

Baixar o Git Bash

Criar um repositório no github (um repositório por projeto)

Copiar o link do repositório

Criar uma pasta local para você colocar seu projeto no github

Clicar com o direito dentro da pasta e **Git Bash Here**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Clonar o projeto usando o link do site usando o seguinte comando:

Git clone (Link)   
Para dar copiar (Shift + Insert)

A partir disso vai ser criado uma pasta com o nome do seu repositório criado no Github

Você irá colar a pasta inteira do seu projeto dentro da pasta clonada

Texto

Descrição gerada automaticamente

E novamente dentro da pasta usar novamente o (Git Bash Here)

O primeiro comando é você adicionar o que você quer que o github armazene

O comando é = Git add .

O “.” significa todos os arquivos de dentro da pasta

Agora você tem que nomear

Git commit –m “nome” e por último colocar lá escrevendo: Git push

**Como atualizar um projeto no GitHub**

Para atualizar um projeto de sua máquina local no GitHub e fazer o controle de versão siga os passos abaixo:

Insira todos os arquivos alterados do projeto no repositório local do Git:

Git add .

Faça o commit do projeto.

O flag -m permite colocar uma frase curta descrevendo a alteração executada no projeto:

Git commit -m "descrição breve do commit"

Envie o projeto local para o branch master no GitHub. Caso esteja trabalhando em outro branch, substitua o nome master pelo nome do branch:

Git push -u origin master

------------------------------------------------------------------------------------------------------------------------------

A quantidade de if e totalmente relacionada a quantidade de respostas possíveis

Salario + salario \* 10 /100

Salario + salario \* 0.10 = 10 /100

Salario \* 1.10

***SWITCH/CASE***

É uma estrutura de condição que define o código a ser executado com base em uma comparação de valores

O comando **break** é utilizado para especificar a última linha de código a ser executada dentro de uma condição. Se declarado, os códigos implementados dentro dos **cases** subsequentes serão executados.

**Default:** o operador default é utilizado quando precisamos definir um fluxo alternativo para as s situações em que o valor contido no switch não seja atendido por nenhum dos  **cases** especificados.

Caso o valor do switch não seja igual a um dos valores contidos nos **cases**, o sistema irá executar o código implementando no bloco **default**; o código 3

***EXERCÍCIO SWITCH/CASE***

Para demonstrar um exemplo de switch/case, considere um programa que informa ao usuário a  **quantidade de dias de um mês** a partir do nome de um mês por ele informado.

**EXERCICIO AVALIATIVA LÓGICA – ESTRUTURA CONDICIONAL**

1 Um determinado clube de futebol pretende classificar seus atletas em categorias e para isto ele contratou um programador para criar um programa que solicite o nome e a idade de um atleta e imprima sua categoria, de acordo com a tabela abaixo:

**De 05 a 10 anos – Infantil**

**De 11 a 15 anos – Juvenil**

**De 16 a 20 anos – Junior**

**De 21 e 25 anos – Profissional**

2 - Escrever um algoritmo que **lê o nome de um funcionário, o número de horas trabalhadas, o valor que recebe por hora** e **o número de filhos**. Com estas informações, **calcular o salário deste funcionário,** sabendo que para cada filho, o funcionário recebe 3% a mais, calculado sobre o salário bruto.

***LISTA DE EXERCÍCIOS CONDICIONAL***

1 - Ler dois valores numéricos inteiros e apresentar o resultado da diferença do maior pelo menor valor. FEITO

2 - Efetuar a leitura de um valor inteiro positivo ou negativo e apresentar o número lido como sendo um valor positivo, ou seja, o programa deverá apresentar o módulo de um número fornecido. Lembre-se de verificar se o número fornecido é menor que zero; sendo, multiplique-o por -1. FEITO

3 – Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir uma mensagem dizendo que o aluno foi aprovado, se o valor da média escolar for maior ou igual a 7. Se o valor da média for menor que 7, solicitar a nota de exame, somar com o valor da média e obter nova média. Se a nova média for maior ou igual a 5, apresentar uma mensagem dizendo que o aluno foi aprovado em exame. Se o aluno não foi aprovado, indicar uma mensagem informando esta condição. Apresentar com as mensagens o valor da média do aluno, para qualquer condição.

FEITO

4 - Efetuar a leitura de três valores e apresentá-los dispostos em ordem crescente.

FEITO

5 – Efetuar a leitura de quatro números inteiros e apresentar os números que são divisíveis por 2 e 3. FEITO

6 - Elaborar um programa que efetue a leitura do nome e do sexo de uma pessoa, apresentando como saída uma das seguintes mensagens: "Ilmo. Sr.", se o sexo informado for masculino, ou a mensagem "Ilma. Sra.”, se o sexo informado for feminino.   
Apresente também junto da mensagem de saudação o nome previamente informado.

FEITO

7 - Ler uma temperatura em graus Celsius e apresentá-la convertida em graus Fahrenheit.   
A fórmula de conversão é F = (9 \* C + 160) / 5, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

FEITO

8 - Ler uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius.   
A fórmula de conversão é C = (F - 32) \* 5/9, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.

FEITO

9 - Elaborar um programa que efetue a apresentação do valor da conversão em real de um valor lido em dólar.   
O programa deve solicitar o valor da cotação do dólar e a quantidade de dólares disponível com o usuário, para que seja apresentado o valor em moeda brasileira.

FEITO

10 - Desenvolva um algoritmo que calcule o IMC de uma determinada pessoa, onde o usuário informa o nome, idade, peso e altura. O cálculo do IMC é feito dividindo o peso (em quilogramas) pela altura (em metros) ao quadrado. Consultar a tabela com os valores de para exibir na tela a mensagem para o usuário.

FEITO

**ESTRUTURA REPETIÇÃO/LAÇO DE REPETIÇÃO/LOOPING – WHILE**

O **while** trata-se de repetição mais utilizada quando programamos com C#. Com ela, enquanto a condição for verdadeira o bloco de código será executado. Primeiramente o sistema testa essa condição. Caso verdadeira, executa as linhas declaradas dento do while; do contrário, sai do loop.

O comando **while** é geralmente utilizado quando não sabemos quantas vezes é necessário repetir um bloco de comandos para que obtenhamos uma condição verdadeira. Esta condição de retorno deve ser booleana.

while (condição)

{  
  
 //Bloco de código  
  
}

**WHILE/DO WHILE/ FOR/FOREACH**

quando precisamos executar um bloco de código repetidas vezes devemos recorrer as estruturas de repetição. Assim, conseguimos programar o código desejado sem que para isso criemos cópias desse mesmo conjunto de instruções. Temos quatro opções para implementas estruturas de repetição; **While, Do While, For e Foreach**.

While dentro do bloco de código criar alguma condição que retorne a condição falsa caso isso não seja feito criaremos um looping infinito.

**ESTRUTURA REPETIÇÃO - DO WHILE**

Esta estrutura de repetição de forma semelhante ao while, porém, ela garante que o código dentro do loop seja executado pelo menos uma vez. Para isso, a condição é declarada após o bloco de código.

do

{  
  
 //Bloco de código  
  
} while (condição);

A diferença entre uma declaração while e do while e que no laço while, a avaliação da expressão logica e feitas antes de se executarem as instruções nele contidas enquanto no laço **do while,** primeiro se executam as instruções e depois realiza-se a avaliação da expressão logica

**ESTRUTURA REPETIÇÃO - FOR**

O  **For**  é a estrutura de repetição que utilizamos quando sabemos a quantidade de repetições que um bloco deve ser executado.

For (inicializador; condição; iterador)  
{  
 // bloco de código  
  
}

A sintaxe é composta por três expressões separadas por ponto e vírgula, início do loop e as condições de como cada iteração irá executar.  
inicializador é uma variável.

**EXERCICIO – LAÇO DE REPETIÇÇÃO – WHILE/ DO WHILE/ FOR**

Criar um loop usando **while, do while,** e **for** que imprimira em um Listbox os valores de 1 a 50.

**ESTRUTURA REPETIÇÃO - BREAK**

Por padrão um loop só e finalizado quando a condição de parada e atendida. Porém e possível interromper a execução de um laço de repetição utilizando o comando break. Esse comando não espera nenhum parâmetro, portanto basta informar a palavra reservada break na linha em que o loop deve ser interrompido.

Int i = 1;  
while ( i < 10)  
{  
 if ( i == 5)  
 break;  
}  
  
além do break há outro comando capaz de modificar o fluxo natural de execução de loop: o continue

***ESTRUTURA REPETIÇÃO - FOREACH***

O **Foreach** e um recurso do C# que possibilita um conjunto de comandos para cada elemento presente em uma coleção (Array, list, stack). Portanto, diferentemente do while e do for, não precisamos definir uma condição de parada. Isso e definido de forma implícita, pelo tamanho da coleção.

Foreach (tipo de elemento in coleção)  
{  
 //BLOCO DE CÓDIGO  
}

*Na declaração do* ***Foreach****, entre parênteses criamos um elemento do tipo utilizado na coleção e, com o operador in, informamos a coleção a ser percorrida, assim, a cada loop os dados presentes em uma posição da coleção são atribuídos ao elemento. `Por fim, entre chaves, inserirmos o código a ser executado no loop.*

*Vetor/matriz posição inicial = posição 0.*

*Vetor/matriz diferenciar cada elemento.  
 virgula para separar cada elemento.*

**Exercícios Estrutura de Repetição**

1 - Apresentar os quadrados dos números inteiros de 15 a 200.

2 - Elaborar um programa que apresente no final o somatório e a média dos valores pares existentes na faixa de 1 até 500.

3 - Elaborar um programa que apresente como resultado o valor do fatorial dos valores ímpares situados na faixa numérica de 1 a 10.

4 - Faça um algoritmo que exiba quantas pessoas possuem mais de 18 anos. O algoritmo deverá ler a idade de 10 pessoas.

if (num % 2 == 0)

continue;

62750 par

Media 251

62500 impar

Media 250