

# **Projeto Final: CIC 133 - Paradigmas de Programação**

**Ivan Leoni Vilas Boas 2018009073, Bruno Brandão Borges 2018014331**

Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)  
Caixa Postal 50 CEP: 37500 903 – Itajubá – MG – Brasil

<ivanleoni30@unifei.edu.br>, <borges.brandao.bruno@unifei.edu.br>

## **1. Introdução**

Um dos motivos de desenvolver novas linguagens de programação está relacionado com o tipo e a natureza de um problema, do qual um programador quer solucionar. Essas novas linguagens podem ser baseadas em alguns dos paradigmas de programação e deixar tal solução um pouco mais fácil, dependendo do tipo de paradigma abordado.

Exemplificamos as soluções de três problemas em 4 paradigmas, e em 4 linguagens diferentes. Os problemas são apresentados abaixo:

- 1) Oito rainhas: O problema consiste em dispor 8 rainhas em um tabuleiro 8x8 de modo que elas não se ataquem. Um ataque é identificado quando há mais de uma rainha na mesma linha, coluna ou diagonal.
- 2) MDC: Calcular o máximo divisor comum entre dois números naturais.
- 3) MMC: Calcular o mínimo múltiplo comum entre dois números naturais.

Partindo deste princípio, este projeto objetiva solucionar os três problemas (Oito rainhas, MMC e MDC) utilizando 4 paradigmas de programação apresentados durante o curso: procedimental, orientado a objetos, lógico e funcional. Para solucionar os problemas foram utilizadas quatro linguagens de programação: C, Java, LISP e PROLOG. Sendo cada uma das linguagens utilizadas no seu tipo de paradigma dominante correspondente.

Na seção dois será discorrido brevemente o histórico com algumas características de cada linguagem utilizada para solucionar os 3 problemas apresentados. Na Seção três será apresentado o ambiente de programação e na seção 4 será fornecida a resolução e a implementação em cada um dos paradigmas: PP, POO, PF e PL. As linguagens escolhidas, respectivamente, foram: C, Java, Lisp e Prolog. Em seguida será realizada às observações acerca das implementações realizadas nos 4 paradigmas e por fim concluir-se-á o estudo praticado e aqui apresentado.

## **2. Breve histórico da linguagem**

Nesta seção será apresentado um breve histórico com algumas características das linguagens utilizadas na execução deste trabalho. Cada linguagem irá representar um paradigma específico. As linguagens aqui descritas são: C, Java, Lisp e Prolog.

## **2.1. Linguagem C:**

A linguagem C foi criada em 1972 e implementada em um computador DEC PDP-11 por Dennis Ritchie no Bell Laboratories. C é derivada de duas outras linguagens: Algol 68 e BCPL.

O foco da linguagem C inicialmente foi o desenvolvimento de sistemas operacionais e compiladores. C foi usada com grande êxito na construção de uma nova versão do sistema operacional Unix, que inicialmente foi escrito em Assembly. O grande sucesso obtido no mundo do Unix fez com que a linguagem ganhasse mais e mais adeptos e atualmente, quase todos os grandes sistemas operacionais são construídos em C/C++.

Além disso, no início dos anos 80, a linguagem passou a ser reconhecida como uma linguagem de propósito geral e contava com diversos compiladores desenvolvidos por vários fabricantes. Nessa época havia uma série de compiladores C semelhantes, porém estes compiladores frequentemente apresentavam várias discrepâncias e eram incompatíveis entre si. Isto tornava a padronização da linguagem uma necessidade real. A padronização foi iniciada pela ANSI (American National Standard Institute) em 1983 e finalizada em 1989. Em 1999 foi aprovada uma revisão do padrão anterior e novas funcionalidades foram acrescentadas. Esta revisão ficou conhecida como C99.

Um grande esforço de padronização foi feito a fim de padronizar a linguagem. Atualmente ainda há versões de compiladores próprias de cada fabricante, porém a maioria dos fabricantes oferece uma opção de compatibilidade com o padrão ANSI C. Em suma, a linguagem C é uma linguagem de propósito geral, o que quer dizer que se adapta a praticamente qualquer tipo de projeto, sendo portátil e com tempo de execução rápido. A linguagem C++ é uma evolução da linguagem C que incorpora orientação a objetos. Vale ressaltar ainda que linguagens como Java e C# foram influenciadas pela linguagem C.

## **2.2. Linguagem Java**

A história começa em 1991, em San Hill Road empresa filiada a Sun (hoje pertencente a Oracle), formado pelo time de engenheiros liderados por Patrick Naughton, Sun Fellow e James Gosling. Os engenheiros da Sun Microsystems iniciaram o projeto “Green Project”, para desenvolver aparelhos domésticos que se comunicassem entre si. A ideia não foi muito para frente devido a limitações e desinteresse, porém gerou a base do que seria o Java. A linguagem de programação chamada de Oak (carvalho) foi criada pelo chefe do projeto James Gosling, que depois decidiram trocar para o nome atual Java.

Em 1995 a Sun viu uma oportunidade na Web, nessa época nas páginas não existia muita interatividade, apenas conteúdos estáticos eram exibidos. Então nesse ano a Sun anunciou o ambiente Java, sendo um absoluto sucesso, gerando uma aceitação aos browsers populares.

O Java foi o primeiro a utilizar decodificadores de televisões interagindo em dispositivos portáteis e outros produtos eletrônicos de consumo, foi do mesmo jeito que foi iniciado em 1991, possuindo portabilidade para qualquer ambiente e do desenvolvimento para múltiplas plataformas, em ambientes de eletrônicos de consumo, desde então o Java vem liderando o mercado em termos de linguagem.

A linguagem foi vendida em 2009 para a gigantesca Oracle.

Entre as principais características e vantagens da linguagem Java podemos destacar: Suporte à orientação a objetos, alta Performance, dinamismo, portabilidade, segurança, simplicidade, distribuído, tipada (detecta os tipos de variáveis quando declaradas) e Interpretada (o compilador pode executar os bytecodes do Java diretamente em qualquer máquina). Dentro das características, o principal item é o fator da Independência de plataforma que possibilitou seu sucesso. Hoje a maioria das linguagens sofrem na transferência de plataforma quando o sistema desenvolvido tem que migrar para outra plataforma, pois quando compilado um programa a ação do compilador é transformar o arquivo-fonte em código de máquina. Já os programas em Java possuem uma característica fundamental que permite desenvolver sem se preocupar com o tipo de sistema ou plataforma que precisa ser desenvolvida e preparada. A independência da plataforma do Java possibilita que o programa seja executado em diferentes plataformas e sistemas operacionais, através de um emulador conhecido como a Máquina Virtual Java ou JVM (Java Virtual Machine) que ajuda a rodar os sistemas baseados em Java. Pode-se também se denominar como uma máquina virtual baseada em software que é executada dentro dos aparelhos eletrônicos onde irá ler e executar os bytecodes do Java.

### **2.3. Linguagem Prolog**

O Prolog(PROgrammation en LOGique) nasceu, na França entre 1971 e 1973 por Philippe Roussel e Alain Colmerauer e seus associados na Universidade de Marseille, com o propósito inicial de traduzir linguagens naturais ao implementar um sistema de comunicação homem-máquina em linguagem natural. Sendo uma linguagem de desenvolvimento prático baseada na ideia de raciocínio dedutivo automatizado. A linguagem permitiu fórmulas para ser interpretada de tal forma que uma conclusão lógica poderia ser alcançada. Assim a linguagem de programação de alto nível baseada em lógica formal, nasceu de um projeto que não tinha por foco a implementação de uma linguagem de programação, mas o processamento de linguagens naturais.

Em 1977, David Warren da Universidade de Edimburgo, implementou uma eficiente versão do Prolog, batizada de Prolog-10. A partir daí, tornou-se uma escolha natural para a resolução de problemas que envolvem a representação simbólica de objetos e relações entre objetos.

Desde 1980 vem influenciando a formação de sistemas inteligentes de computação, lógica de programação e aprendizagem de máquina. Atualmente o Prolog tem alta aplicação na área de Inteligência Artificial, sendo utilizado em diversas aplicações na área de computação simbólica, incluindo em bases de dados relacionais, sistemas especialistas, lógica matemática, prova automática de teoremas, resolução de problemas abstratos e geração de planos, processamento de linguagem natural, projeto de arquiteturas, logística, resolução de equações simbólicas, construção de compiladores, análise bioquímica e projeto de fármacos.

O Prolog é uma linguagem de programação prática e eficiente. O base do Prolog é a programação em lógica, onde o processo de computação pode ser visto como uma sequência lógica de inferências, para descrição e resolução de problemas. O Prolog é

uma linguagem diferente, mas de uma simplicidade marcante. Essa diferença decorre do fato de o Prolog ser uma linguagem funcional, e não uma linguagem procedural. Sua estrutura conceitual está intimamente ligada com a lógica matemática, o que a torna uma ferramenta indispensável para o estudo de lógica. Versões diferentes do Prolog podem ser encontradas nas mais diversas plataformas, tanto na forma de compiladores como de interpretadores.

Prolog é uma linguagem interativa que permite resolver problemas que envolvem representação simbólica de objetos e seus relacionamentos. O advento da linguagem Prolog reforçou a tese de que a lógica é um formalismo conveniente para representar e processar conhecimento. Prolog evita que o programador descreva procedimentos para obter a solução de um problema, permitindo que ele expresse declarativamente apenas a estrutura lógica do problema através de termos, fórmulas atômicas e cláusulas.

Entre as principais características da linguagem Prolog podemos citar: representa uma implementação da lógica como linguagem de programação, orientada para processamento simbólico, apresenta uma semântica declarativa inerente à lógica, permite a obtenção de respostas alternativas, suporta estrutura de dados que permite simular registros ou listas, permite recuperação dedutiva de informação, representa programas e dados através do mesmo formalismo (cláusulas), incorpora facilidades computacionais extra, permite construir protótipos rapidamente, elegância e facilidade de compreensão, linguagem de 5ª geração, modularidade, polimorfismo, alocação e desalocação automática de memória dinâmica, compilação incremental, metaprogramação (programas que manipulam outros programas) e por fim o Prolog é uma linguagem declarativa, isto é, o usuário "declara" o modelo do seu problema e deixa o Prolog buscar a solução.

## **2.4. Linguagem Lisp**

A Lisp é uma família de linguagens que possui uma longa história. As primeiras ideias para a linguagem foram desenvolvidas por John McCarthy em 1956, durante um projeto de pesquisa em inteligência artificial. A primeira implementação da linguagem se dá no inverno de 1958. A Lisp foi desenvolvida como uma ferramenta matemática e se tornou uma família de linguagens, assim como C e C++. A linguagem usa funções que representam expressões matemáticas.

Durante as décadas de 80 e 90, se destacaram dois dialetos LISP, Common LISP e o Scheme. O Scheme trouxe várias características que deixaram os códigos mais confiáveis. No final dos anos 70, vários tipos de LISP estavam em uso e então houve uma união com o intuito de incluir todas as vantagens que cada tipo oferecia e amenizar as desvantagens. Nas décadas de 70 e 80 o Lisp foi amplamente usado pela comunidade em aplicações de inteligência artificial. Se tornou a principal linguagem da comunidade de inteligência artificial, tendo sido pioneiro em aplicações como administração automática de armazenamento, linguagens interpretadas e programação funcional. Atualmente o Lisp é usado em alguns projetos importantes, como a linguagem de extensão do software do AutoCAD.

Linguagens funcionais, como o LISP enfatizam o processo de identificar blocos

e partes repetidas de código (como a leitura de arquivo ou o cálculo de uma raiz) e constróem funções que encapsulam a funcionalidade dentro de uma simples definição. Isto aumenta a manutenibilidade (a definição de uma rotina é feita uma única vez no programa) e a confiabilidade (não ocorrem erros de tipagem no caso de outras definições da mesma rotina).

Principalmente devido ao seu longo tempo de existência, LISP é uma linguagem funcional atípica, pois também suporta muitas das estruturas das linguagens imperativas, estruturas essas ausentes nas modernas linguagens de funcionais. Entre as características da linguagem Lisp podemos citar:

- Fraca Tipagem: LISP, em relação a outras linguagens funcionais mais recentes, é fracamente tipado, o que causa complicações, já que operações que acessam as suas estruturas de dados são tratadas como funções.
- Um alto nível de abstração, especialmente quando as funções são utilizadas, suprimindo muitos detalhes da programação e minimizando a probabilidade da ocorrência de muitas classes de erros;
- Funções de ordem elevada: Linguagens funcionais tipicamente suportam funções de ordem elevada (exemplo: função de uma função de uma função de uma...)
- Concorrência (multitarefa): A linguagem funcional intrinsecamente nos oferece oportunidades para a concorrência: A partir do momento em que uma função tem mais de um parâmetro, estes parâmetros devem em princípio ser avaliados simultaneamente (note que os parâmetros seriam as funções correspondentes às tarefas a serem executadas); A partir deste ponto, a responsabilidade pela sincronização das tarefas passa do programador para o compilador (as modernas linguagens funcionais orientadas a multitarefa dispõem de mecanismos através dos quais o programador pode guiar o compilador). Todavia, as linguagens funcionais orientadas a multitarefa permitem ao programador trabalhar em um nível muito mais elevado do que as linguagens imperativas destinadas a este mesmo fim.
- Avaliação Ociosa: É o que ocorre quando uma função aninhada executa uma computação desnecessária para a avaliação da função que a chama, aumentando o tempo de execução.
- A não dependência das operações de atribuição permite aos programas avaliações nas mais diferentes ordens. Esta característica de avaliação independente da ordem torna as linguagens funcionais as mais indicadas para a programação de computadores maciçamente paralelos;
- A ausência de operações de atribuição torna os programas funcionais muito mais simples para provas e análises matemáticas do que os programas procedurais.
- Como desvantagem, destaca-se uma menor eficiência para resolver problemas que envolvam muitas variáveis ou muitas atividades sequenciais que são muitas vezes mais fáceis de se trabalhar com programas procedurais ou programas orientados a objeto.

A variação do Lisp que será utilizado neste trabalho é o Common Lisp, que é uma implementação específica da linguagem de programação Lisp multi-paradigma que suporta programação funcional e procedural. Sua especificação foi criada por Guy L. Steele nos anos 1980 a partir da linguagem Lisp com o intuito de combinar aspectos de diversos dialetos Lisp anteriores, incluindo Scheme. Foi proposto inicialmente o nome

de "Standard Lisp" para a linguagem, mas em virtude de um dialecto Lisp já existir com este nome, se buscou um nome similar, resultando no nome "Common Lisp".

Em 1994 foi publicada no padrão ANSI o Common Lisp que é bem maior e semanticamente mais complexa que Scheme uma vez que foi projetada para ser uma linguagem comercial e ser compatível com os diversos dialectos Lisp dos quais derivou.

O Common Lisp permite várias representações diferentes de números. Estas representações podem ser divididas em 4 tipos: hexadecimais, octais, binários e decimais. Estes últimos podem ser divididos em 4 categorias: inteiros, racionais, pontos flutuantes e complexos.

### 3. Ambiente de programação

Os ambientes de programação utilizado para a resolução, ou seja, para a codificação e execução dos três problemas apresentados será descrito a seguir:

O ambiente de desenvolvimento para o paradigma imperativo em C e Java: Foi utilizado **Visual Studio Code** versão 1.62.0 para codificação, mas para executar foi utilizado o **Powershell** do Windows 10 V.21H1. Para o paradigma Funcional na linguagem Common Lisp foi utilizado o **GNU Common Lisp** versão 2.6.2 e para o paradigma lógico em Prolog foi utilizado o **SWI-Prolog** versão 8.5.1-7-gf3b79d662.

Paradigma	Language m	Ambiente de Programação	Versão
PP	C	Visual Studio Code e Powershell	1.62.0
POO	Java		e Windows 21H1
PF	Common Lisp	GNU Common Lisp	2.6.2
PL	Prolog	SWI-Prolog	8.5.1-7

*Tabela 01: Ambiente de Programação*

### 4. Descrição da solução e da implementação para cada problema

A seguir serão apresentados as soluções e implementações a cerca dos 3 problemas (Oito rainhas, MMC e MDC) nos 4 paradigmas procedimental (PP), orientado a objetos (POO), lógico (PL) e funcional (PF) com as respectivas linguagens: C, Java, Prolog e Lisp.

## 4.1. Solução da Oito Rainhas para PP com a Linguagem C

A solução do problema das 8 Rainhas no PP na linguagem contempla a utilização de duas funções e de um procedimento, além da main(). São elas:

- naoEstaAmeacada(): esta função faz a análise das posições do tabuleiro para verificar conforme as regras a melhor posição para colocar a rainha. Com o uso dos 3 laços é verificado a presença de uma rainha na coluna ou nas diagonais, caso não encontre a rainha em nenhuma das posições de ataque, será então retornada a posição ideal para inserir a nova rainha.
- colocarRainha(): Esta função determinará de forma recursiva as posições exatas das 8 rainhas do tabuleiro 8 x 8 quando encontrada a posição ideal esta será identificada como TRUE.
- apresentarTabuleiro(): Este procedimento irá apresentar a solução, colocando 'R' nas posições ideais das rainhas que foram determinadas pelas duas funções anteriores e irá atribuir 'X' às posições vagas.

Além dessas funções, foi necessário a função main para a chamada e execução dessas funções. A seguir será detalhado melhor cada função e apresentado os códigos em partes para melhor entendimento.

A primeira função naoEstaAmeacada() consiste em visitar cada posição do tabuleiro e verificar se a posição está livre, ou seja, se ela não representa nenhum perigo para a rainha, da qual não pode estar na mesma diagonal, lateral ou na mesma coluna que uma outra rainha (verificado pelos laços 3 laços de repetição). Esta função utiliza 3 parâmetros de entrada: o tabuleiro e a posição que é dada pela linha e a coluna. Se a posição analisada for boa para colocar a rainha, a mesma então retornará TRUE, caso a posição fornecida seja ameaçadora para ataque será então retornado FALSE. Logo abaixo apresentamos o código desta função:

```
int naoEstaAmeacada(int tabuleiro[8][8], int linha, int coluna){
    int i, j, posicaoBoa = TRUE;
    i = linha - 1;
    while (i >= 0 && posicaoBoa){ // verifica coluna
        posicaoBoa= !tabuleiro[i][coluna];
        i = i - 1;
    }
    i = linha - 1;
    j = coluna + 1;
    while (i >= 0 && j < 8 && posicaoBoa){
        //verifica diagonal secundaria
        posicaoBoa= !tabuleiro[i][j];
        i--;
        j++;
    }
}
```

```

        i = linha - 1;
        j = coluna - 1;
        while (i >= 0 && j >= 0 && posicaoBoa){
            //Verifica diagonal principal
            posicaoBoa = !tabuleiro[i][j];
            i--;
            j--;
        }
        return posicaoBoa;
    }
}

```

A segunda função `colocarTabuleiro()` tem como finalidade dispor as próprias rainhas no tabuleiro de modo que a função “`naoEstaAmeaçada`” seja chamada para verificar se esta posição não representa um perigo para a Rainha. Portanto, esta função vai colocar as rainhas nos seus devidos lugares deixando a posição com valor `TRUE` como conteúdo. Os parâmetros de entradas necessários para esta função são o tabuleiro e a linha e retorna `TRUE` caso as 8 rainhas sejam colocadas conforme regra definida, caso não seja possível rearranjar as rainhas seria então retornado `FALSE`. Logo abaixo apresentamos o código desta função:

```

int colocarRainha(int tabuleiro[8][8], int linha){
    int coluna = 0, posicao = FALSE;
    if (linha >= 8){ //Parada da recursividade
        return TRUE;
    }
    else{
        while (coluna < 8 && !posicao){
            tabuleiro[linha][coluna] = TRUE;
            if (naoEstaAmeaçada(tabuleiro, linha, coluna)){
                //Verifica se a posição é apropriada
                posicao = colocarRainha(tabuleiro, linha + 1);
            }
            if (!posicao){ //se a posição não for apropriada
                tabuleiro[linha][coluna] = FALSE;
                coluna = coluna + 1;
            }
        }
        return posicao;
    }
}
}

```



O terceiro procedimento denominado de apresentarTabuleiro, possui como principal finalidade apresentar primeiramente o tabuleiro vazio e depois da realocação das rainhas nas posições ideais representá-lo com a disposição das rainhas. A letra “X” significa os lugares(posições) vazios do tabuleiro e o caractere “R” representa as próprias rainhas dispostas conforme a solução encontrada. Assim, as posições do tabuleiro que apresentam valor TRUE são apresentadas como “R” e as posições atribuídas ao valor FALSE são apresentadas como “X”. O procedimento exige apenas como parâmetro o próprio tabuleiro para sua impressão. A seguir é apresentado as linhas de códigos para esta implementação.

```
void apresentarTabuleiro(int tabuleiro[8][8]){
    int i, j;
    for (i = 0; i < 8; i++){
        for (j = 0; j < 8; j++){
            if (tabuleiro[i][j]){
                printf("  R");
            }
            else{
                printf("  X");
            }
        }
        printf("\n\n");
    }
}
```

A seguir, por fim, é apresentada a função principal, que inicialmente inicia o tabuleiro com todas as posições iguais à FALSE, apresentando, assim, o tabuleiro vazio. Em seguida é realizada a chamada a função colocarRainha(), esta por sua vez faz uso da função naoEstaAmeaçada() e assim ambas estabelecem as futuras posições das 8 rainhas no tabuleiro. Se reajandas as rainhas corretamente quando retornado a main() então é reapresentado o tabuleiro com as rainhas nas posições adequadas, caso contrário uma mensagem de não realocamento das rainhas será emitida. A função main() pode ser observado a seguir:

```
int main(void){
    int i, j, tabuleiro[8][8];
    printf("  Criação Tabuleiro 8X8\n\n");
    for (i = 0; i < 8; i++){
        for (j = 0; j < 8; j++){
            tabuleiro[i][j] = FALSE;
        }
    }
}
```

```

    apresentarTabuleiro(tabuleiro);

    if (colocarRainha(tabuleiro,0)){
        printf("\n Distribuição da Rainha tabuleiro 8X8\n\n");
        apresentarTabuleiro(tabuleiro);
    }
    else{
        printf("\n Rainha não pode ser colocada.\n");
    }
    getchar ( );
    return 0;
}

```

Vale ressaltar que foi só utilizado a biblioteca C stdio.h que apresenta as funções básicas de c, como printf, por exemplo. E por fim, como a linguagem C não apresenta tipo booleano foi utilizado e definido duas constantes para facilitar no desenvolvimento. Assim as posições definidas como FALSE na verdade recebem 0 como conteúdo armazenado na memória e as posições definidas como TRUE recebem 1.

```

#include <stdio.h>
#define FALSE 0
#define TRUE 1

```

A Seguir é demonstrado a saída do código, ou seja, uma das possíveis soluções para este problema das 8 rainhas em C.

```
C:\Users\IVAN LEONI\Desktop\8 Rainhas\C\rainha8.exe
Criacao Tabuleiro 8X8
X X X X X X X X
X X X X X X X X
X X X X X X X X
X X X X X X X X
X X X X X X X X
X X X X X X X X
X X X X X X X X
X X X X X X X X

Distribuicao da Rainha tabuleiro 8X8
R X X X X X X X
X X X X R X X X
X X X X X X X R
X X X X X R X X
X X R X X X X X
X X X X X X R X
X R X X X X X X
X X X R X X X X
```

## 4.2. Solução da Oito Rainhas para POO com a Linguagem Java

A solução do problema das oito rainhas usando o paradigma orientado a objetos em Java utiliza a mesma lógica que a solução em C. São utilizadas as mesmas funções, aqui denominadas de métodos. São eles `colocarRainha()`, `naoEstaAmeacada()` e `apresentarTabuleiro()`. Porém são alteradas algumas ações na linguagem, como por exemplo utilizando `System.out.print` para imprimir a solução do problema. Também é utilizada a variável booleana, não presente na linguagem C, e que facilita a realização de comparações. Foi instanciada uma única classe “App” que representa todo o problema que é composta pelos métodos e a `main()` para a solução do problema.

Por se tratar do paradigma imperativo, com variação de estados das variáveis, os métodos são muitos similares às funções usadas na resolução do problema no paradigma Procedimental, uma vez que a maneira (a lógica) utilizada para a resolução é a mesma. Assim são utilizados o método `colocarRainha()` para colocar uma rainha em uma casa do tabuleiro de forma a seguir a regra, onde a regra é definida pelo outro método `naoEstaAmeacada()` que faz a verificação da existência de possíveis conflitos e um outro método `apresentarTabuleiro()` para mostrar a disposição inicial e final do tabuleiro com uma possível solução para o problema, além o procedimento principal (`main`), responsável pela chamada inicialização do código.

O método principal inicializa uma matriz com valores booleanos, e preenche inicialmente todos os elementos da matriz com valor falso. Após isso, é feita a chamada ao método colocarRainha() contendo a matriz tabuleiro[][] como argumento para o primeiro parâmetro e o valor zero (0) para o segundo parâmetro representando a linha inicial do tabuleiro. Caso as rainhas sejam posicionadas corretamente seguindo as regras, no final será impresso o tabuleiro com as rainhas rearranjadas. Conforme pode ser observado no código abaixo:

```
public static void main(String[] args) {
    int i, j;
    boolean tabuleiro[][] = new boolean[8][8];
    System.out.print(" Criacao Tabuleiro 8X8\n\n");
    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            tabuleiro[i][j] = false;
        }
    }
    apresentarTabuleiro(tabuleiro);
    // Se a 8 rainhas forem colocadas conforme as regras
    // problema) no tabuleiro 8x8
    if (colocarRainha(tabuleiro, 0)){
        System.out.print("\nDistribuicao da Rainha no tabuleiro 8X8\n\n");
        apresentarTabuleiro(tabuleiro);
    }
    else {
        System.out.print("\n Rainha não rearranjada no tabuleiro 8x8.\n");
    }
}
```

O método `posicionarRainha()` é responsável por colocar uma rainha no tabuleiro. Nele é utilizado a variável `posição` e `coluna`, que representa a atual coluna que está sendo analisada no tabuleiro e apresenta um laço condicional, que enquanto o valor da coluna for menor que 8 e não existir uma posição possível para colocar uma rainha, será feita a análise da existência de conflitos, para assim encontrar posição ideal a ser inserida a rainha no tabuleiro. Caso não existam erros no momento de posicionar uma rainha no tabuleiro, ela será colocada na posição definida. O código do método `posicionarRainha()` pode ser analisado a seguir:

```
// Posicionar a rainha no tabuleiro 8X8 conforme as regras
public static boolean colocarRainha(boolean tabuleiro[][], int linha) {
    int coluna = 0;
```

```

boolean posicao = false;
if (linha >= 8) { // parada da recursividade
    return true;
}
else {
    while (coluna < 8 && !posicao) {
        tabuleiro[linha][coluna] = true; // coloca rainha
        // Verificar se a peça não está sendo ameaçada
        if (naoEstaAmeacada(tabuleiro, linha, coluna)) {
            posicao = colocarRainha(tabuleiro, linha + 1);
        }
        if (!posicao) { // se rainha ameaçada
            tabuleiro[linha][coluna] = false; // retira rainha
            coluna++; // passa para próxima coluna
        }
    }
    return posicao;
}
}

```

O método `colocarRainha()` realiza a chamada ao `naoEstaAmeacada()` que tem como objetivo analisar a posição escolhida para a rainha ao verificar se existe algum ataque sendo feito à posição da nova rainha. Esse procedimento recebe como parâmetro o tabuleiro, a linha e a coluna da nova rainha e realiza através de 3 laços de repetição a análise para verificar se existe algum ataque na linha, coluna ou diagonal. Se a posição analisada for boa para colocar a rainha será retornado `true`, caso contrário `false`. O código deste método está disposto a seguir:

```

public static boolean naoEstaAmeacada(boolean tabuleiro[][],
                                     int linha, int coluna) {

    int i, j;
    boolean posicaoBoa = true;

    i = linha - 1;
    while (i >= 0 && posicaoBoa) {
        posicaoBoa = !tabuleiro[i][coluna];
        i--;
    }

    i = linha - 1;

```

```

        j = coluna + 1;
        while (i >= 0 && j < 8 && posicaoBoa) {
            posicaoBoa = !tabuleiro[i][j];
            i--;
            j++;
        }

        i = linha - 1;
        j = coluna - 1;
        while (i >= 0 && j >= 0 && posicaoBoa) {
            posicaoBoa = !tabuleiro[i][j];
            i--;
            j--;
        }
        return posicaoBoa;
    }
}

```

Por fim, o método `apresentarTabuleiro` recebe a matriz de tabuleiro e possui como a principal finalidade apresentar o tabuleiro vazio ou com a disposição das 8 rainhas com as demais posições vagas, do qual a letra “X” significa os lugares do tabuleiro em que está vago (vazio) e o caractere “R” representa as próprias rainhas dispostas nas ordens já solucionadas. O código utilizado neste método pode ser observado abaixo:

```

public static void apresentarTabuleiro(boolean tabuleiro[][]) {
    int i, j;
    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            if (tabuleiro[i][j]) {
                // Rainha
                System.out.print("  R");
            } else { // demais posições vagas do tabuleiro
                System.out.print("  X");
            }
        }
        System.out.print("\n\n");
    }
}

```

Vale ressaltar que o tamanho do código e a estrutura da solução do problema para o paradigma imperativo (PP e POO) tanto na linguagem C como Java foi relativamente longa, porém foi fácil compreensão.

### 4.3. Solução da Oito Rainhas para PL com a Linguagem Prolog

Conforme será mostrado a seguir no código-fonte um conjunto de recursos da linguagem Prolog deve ser utilizado para prover a solução do problema das 8 Rainhas.

Inicialmente, observa-se a manipulação de listas para armazenar os pares de coordenadas XY do tabuleiro de xadrez. Operadores aritméticos foram utilizados para a composição de expressões envolvendo as coordenadas XY, para obtenção das posições válidas para o posicionamento de cada uma das oito rainhas.

Alguns Recursos de formatação de saída foram utilizados para desenhar um tabuleiro de xadrez na interface, permitindo a visualização gráfica das posições geradas como solução do problema.

Os predicados recursivos foram implementados para permitir a "busca cega" na lista de coordenadas e montar a lista que apresenta a solução para o problema. Observa-se, ainda, a presença do operador / que permite a combinação dos valores X e Y para a formação de um par de coordenadas.

Em termos da estratégia para resolução do problema, pode-se observar o seguinte:

- Inicialmente, uma lista é construída para armazenar os valores válidos para as coordenadas Y.
- O predicado recursivo **membro** tem a função de verificar se uma determinada coordenada faz parte da lista de coordenadas Y;
- O predicado recursivo **naopodeatacar** gera, a partir de um par de coordenadas XY, uma lista de coordenadas que contém posições em que uma determinada rainha não pode ser atacada;
- os predicados **rainha8...rainha1** são utilizados para gerar a representação gráfica do posicionamento de cada rainha no tabuleiro;
- Os predicados **tabelainferior** e **linhadivisor** são utilizados para formatar a interface do tabuleiro de xadrez;
- O predicado **tabuleiro** apresenta a solução gráfica para o problema;
- O predicado **resolveproblema** aciona a chamada do predicado **solucaonrainhas** que, recursivamente, gera uma lista de coordenadas que define a solução para o problema.

A Seguir é fornecido todo o código utilizado para resolução do problema das 8 rainhas em Prolog:

```
solucaonrainhas([]).
```

```
solucaonrainhas([X/Y | OUTRASCOORDENADAS]) :-  
solucaonrainhas(OUTRASCOORDENADAS),  
membro(Y,[1,2,3,4,5,6,7,8]),
```

*naopodeatacar(X/Y,OUTRASCOORDENADAS).*

*naopodeatacar(\_,[ ]).*

*naopodeatacar(X/Y,[X1/Y1\OUTRASCOORDENADAS]) :-*  
*Y=\=Y1,*  
*Y1-Y=\=X1-X,*  
*Y1-Y=\=X-X1,*  
*naopodeatacar(X/Y,OUTRASCOORDENADAS).*

*membro(X,[X\LISTA]).*  
*membro(X,[Y\LISTA]) :- membro(X,LISTA).*

*linhadivisoria :- write(' +-----+'),nl.*

*tabelainferior :-*  
*write(' +-----+'),nl,*  
*write(' 1 2 3 4 5 6 7 8 '),nl.*

*rainha8(X,[X\LISTA]) :- X == 1/8,write('8|R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha8(X,[X\LISTA]) :- X == 2/8,write('8|\\R\|\\\|\\\|\\\|'),nl.*  
*Rainha8(X,[X\LISTA]) :- X == 3/8,write('8|\\|\\R\|\\\|\\\|\\\|'),nl.*  
*rainha8(X,[X\LISTA]) :- X == 4/8,write('8|\\\|\\\R\|\\\|\\\|'),nl.*  
*rainha8(X,[X\LISTA]) :- X == 5/8,write('8|\\\|\\\|\\R\|\\\|\\\|'),nl.*  
*rainha8(X,[X\LISTA]) :- X == 6/8,write('8|\\\|\\\|\\\|\\R\|\\\|'),nl.*  
*rainha8(X,[X\LISTA]) :- X == 7/8,write('8|\\\|\\\|\\\|\\\|\\R\|\\\|'),nl.*  
*rainha8(X,[X\LISTA]) :- X == 8/8,write('8|\\\|\\\|\\\|\\\|\\\|\\R\|\\\|'),nl.*  
*rainha8(X,[Y\LISTA]) :- rainha8(X,LISTA).*  
*rainha7(X,[X\LISTA]) :- X == 1/7,write('7|R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 2/7,write('7|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 3/7,write('7|\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 4/7,write('7|\\\|\\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 5/7,write('7|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 6/7,write('7|\\\|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 7/7,write('7|\\\|\\\|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[X\LISTA]) :- X == 8/7,write('7|\\\|\\\|\\\|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha7(X,[Y\LISTA]) :- rainha7(X,LISTA).*

*rainha6(X,[X\LISTA]) :- X == 1/6,write('6|R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 2/6,write('6|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 3/6,write('6|\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 4/6,write('6|\\\|\\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 5/6,write('6|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 6/6,write('6|\\\|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 7/6,write('6|\\\|\\\|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[X\LISTA]) :- X == 8/6,write('6|\\\|\\\|\\\|\\\|\\\|\\R\|\\\|\\\|\\\|\\\|'),nl.*  
*rainha6(X,[Y\LISTA]) :- rainha6(X,LISTA).*



*rainha5*(X,[X|LISTA]) :- X == 1/5,write('5|R\AAAAAAAA'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 2/5,write('5|AR\AAAAAAAA'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 3/5,write('5|AAAR\AAAAAAAA'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 4/5,write('5|AAAAAR\AAAA'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 5/5,write('5|AAAAAR\AAAA'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 6/5,write('5|AAAAAR\AA'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 7/5,write('5|AAAAAR\A'),nl.  
*rainha5*(X,[X|LISTA]) :- X == 8/5,write('5|AAAAAR'),nl.  
*rainha5*(X,[Y|LISTA]) :- *rainha5*(X,LISTA).

*rainha4*(X,[X|LISTA]) :- X == 1/4,write('4|R\AAAAAAAA'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 2/4,write('4|AR\AAAAAAAA'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 3/4,write('4|AAAR\AAAA'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 4/4,write('4|AAAAAR\AA'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 5/4,write('4|AAAAAR\A'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 6/4,write('4|AAAAAR'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 7/4,write('4|AAAAAR'),nl.  
*rainha4*(X,[X|LISTA]) :- X == 8/4,write('4|AAAAAR'),nl.  
*rainha4*(X,[Y|LISTA]) :- *rainha4*(X,LISTA).

*rainha3*(X,[X|LISTA]) :- X == 1/3,write('3|R\AAAAAAAA'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 2/3,write('3|AR\AAAAAAAA'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 3/3,write('3|AAAR\AAAA'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 4/3,write('3|AAAAAR\AA'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 5/3,write('3|AAAAAR\A'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 6/3,write('3|AAAAAR'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 7/3,write('3|AAAAAR'),nl.  
*rainha3*(X,[X|LISTA]) :- X == 8/3,write('3|AAAAAR'),nl.  
*rainha3*(X,[Y|LISTA]) :- *rainha3*(X,LISTA).

*rainha2*(X,[X|LISTA]) :- X == 1/2,write('2|R\AAAAAAAA'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 2/2,write('2|AR\AAAAAAAA'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 3/2,write('2|AAAR\AAAA'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 4/2,write('2|AAAAAR\AA'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 5/2,write('2|AAAAAR\A'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 6/2,write('2|AAAAAR'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 7/2,write('2|AAAAAR'),nl.  
*rainha2*(X,[X|LISTA]) :- X == 8/2,write('2|AAAAAR'),nl.  
*rainha2*(X,[Y|LISTA]) :- *rainha2*(X,LISTA).

*rainha1*(X,[X|LISTA]) :- X == 1/1,write('1|R\AAAAAAAA'),nl.  
*rainha1*(X,[X|LISTA]) :- X == 2/1,write('1|AR\AAAAAAAA'),nl.  
*rainha1*(X,[X|LISTA]) :- X == 3/1,write('1|AAAR\AAAA'),nl.  
*rainha1*(X,[X|LISTA]) :- X == 4/1,write('1|AAAAAR\AA'),nl.  
*rainha1*(X,[X|LISTA]) :- X == 5/1,write('1|AAAAAR\A'),nl.  
*rainha1*(X,[X|LISTA]) :- X == 6/1,write('1|AAAAAR'),nl.

```

rainha1(X,[X\LISTA]) :- X == 7/1,write('1\\\\\\\\\\\\\\\\R\\\\\\\\'),nl.
rainha1(X,[X\LISTA]) :- X == 8/1,write('1\\\\\\\\\\\\\\\\\\\\R\\\\\\\\'),nl.
rainha1(X,[Y\LISTA]) :- rainha1(X,LISTA).

```

```

tabuleiro(S) :-
nl,
write('Representacao Grafica para o Problema das Oito Rainhas'),
nl,
write('-----'),
nl,
linhadivisoria,
rainha8(Y8,S),
linhadivisoria,
rainha7(Y7,S),
linhadivisoria,
rainha6(Y6,S),
linhadivisoria,
rainha5(Y5,S),
linhadivisoria,
rainha4(Y4,S),
linhadivisoria,
rainha3(Y3,S),
linhadivisoria,
rainha2(Y2,S),
linhadivisoria,
rainha1(Y1,S),
tabelainferior,
nl,nl,write('Lista contendo a solucao: '),nl.

```

```

imprimeresposta([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).

```

```

resolveproblema(S) :-
imprimeresposta(S),
solucaonrainhas(S),
tabuleiro(S),!.

```

Considerando-se a implementação feita, o seguinte caso de teste foi realizado: **resolveproblema(S)** no **SWI-Prolog**. Como resultado do caso de teste, o programa gera uma lista de pares de coordenadas, lista esta que pode ser decomposta pelos predicados acionados pelo predicado **tabuleiro** para gerar a representação gráfica. A lista de resolução apresenta respectivamente a coluna e linha onde a Rainha R deve se localizar de modo que não seja atacada. Abaixo é apresentada a saída do SWI-Prolog.

SWI-Prolog (AMD64, Multi-threaded, version 8.5.1)

File Edit Settings Run Debug Help

```
?-
|   resolveproblema(S).
```

Representacao Grafica para o Problema das Oito Rainhas

```
-----
+-----+
8|/|/|/|/|/R|/|/|
+-----+
7|/|/|R|/|/|/|/|
+-----+
6|/|/|/|/|R|/|/|
+-----+
5|/|/|/|/|/|R|/|
+-----+
4|R|/|/|/|/|/|/|
+-----+
3|/|/|/|R|/|/|/|
+-----+
2|/|R|/|/|/|/|/|
+-----+
1|/|/|/|/|/|/|R|
+-----+
  1  2  3  4  5  6  7  8
```

Lista contendo a solucao:

```
S = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1].
```

```
?- 
```

#### 4.4. Solução da Oito Rainhas para PF com a Linguagem Lisp

Partindo dos principais conceitos a respeito da linguagem Lisp, a solução do problema das 8 Rainhas utilizando o paradigma funcional utiliza-se da variação do dialeto de Lisp, determinado pelo Common Lisp, a implementação deste problema possui 3 funções principais para a resolução do problema. A seguir será apresentado o código e detalhadas as principais funções utilizadas para a solução.

```
(defun queens (n &optional (m n))

  (if (zerop n)

      (list nil)

      (loop for solution in (queens (1- n) m)

        nconc (loop for new-col from 1 to m

          when (loop for row from 1 to n

            for col in solution

              always (/= new-col col (+ col row) (- col row)))

            collect (cons new-col solution))))))
```

```

(defun print-solution (solution)

(loop for queen-col in solution

  do (loop for col from 1 to (length solution)

    do (write-char (if (= col queen-col) #\Q #\.))))

  (terpri))

(terpri))

(defun print-queens (n)

(mapc #'print-solution (queens n)))

```

A primeira função `queens()` consiste em determinar as posições das rainhas de forma que elas respeitem as posições para não haver ataques, definido assim uma forma de recolocar as 8 rainhas na posições ideais. Para isso é feito um laço de repetição, denominado pelo `loop`, de modo que este percorre todas as linhas e colunas do tabuleiro e vai dispondo a rainhas nas suas devidas posições.

A função `print-solution()` é utilizada para imprimir e retornar a posição das rainhas no tabuleiro de modo que elas respeitem as regras.

A função `print-queens()` realiza a chamada a função `print-solution()` para mostrar o tabuleiro com as rainhas nas suas devidas posições.

A codificação completa no GNU Common Lisp com todas as funções podem ser observadas a seguir:

```
C:\PROGRA~1\GCL-2.6.2-ANSI\lib\gcl-2.6.2\unixport\saved_ansi_gcl.exe
GCL (GNU Common Lisp) 2.6.2 ANSI Jun 28 2004 15:27:28
Source License: LGPL(gcl,gmp), GPL(unexec,bfd)
Binary License: GPL due to GPL'ed components: (UNEXEC)
Modifications of this banner must retain notice of a compatible license
Dedicated to the memory of W. Schelter

Use (help) to get some basic information on how to use GCL.

>(defun queens (n &optional (m n))
  (if (zerop n)
      (list nil)
      (loop for solution in (queens (1- n) m)
            nconc (loop for new-col from 1 to m
                      when (loop for row from 1 to n
                                for col in solution
                                always (/= new-col col (+ col row) (- col row)))
                    collect (cons new-col solution))))))

QUEENS

>
(defun print-solution (solution)
  (loop for queen-col in solution
        do (loop for col from 1 to (length solution)
                  do (write-char (if (= col queen-col) #\Q #\.)))
        (terpri))
  (terpri))

PRINT-SOLUTION

>
(defun print-queens (n)
  (mapc #'print-solution (queens n)))

PRINT-QUEENS
```

Vale ressaltar que o tamanho do código e a estrutura da solução do problema nesta linguagem foi muito precisa e curta, denotando uma simplicidade em termos de escrita, mas sua legibilidade é bem complexa para os iniciantes da linguagem. A resolução completa das soluções é apresentada a seguir:

```
C:\PROGRA~1\GCL-2.6.2-ANSI\lib\gcl-2.6.2\unixport\saved_ans...
PRINT-QUEENS
>(QUEENS 8)

((4 2 7 3 6 8 5 1) (5 2 4 7 3 8 6 1) (3 5 2 8 6 4 7 1)
 (3 6 4 2 8 5 7 1) (5 7 1 3 8 6 4 2) (4 6 8 3 1 7 5 2)
 (3 6 8 1 4 7 5 2) (5 3 8 4 7 1 6 2) (5 7 4 1 3 8 6 2)
 (4 1 5 8 6 3 7 2) (3 6 4 1 8 5 7 2) (4 7 5 3 1 6 8 2)
 (6 4 2 8 5 7 1 3) (6 4 7 1 8 2 5 3) (1 7 4 6 8 2 5 3)
 (6 8 2 4 1 7 5 3) (6 2 7 1 4 8 5 3) (4 7 1 8 5 2 6 3)
 (5 8 4 1 7 2 6 3) (4 8 1 5 7 2 6 3) (2 7 5 8 1 4 6 3)
 (1 7 5 8 2 4 6 3) (2 5 7 4 1 8 6 3) (4 2 7 5 1 8 6 3)
 (5 7 1 4 2 8 6 3) (6 4 1 5 8 2 7 3) (5 1 4 6 8 2 7 3)
 (5 2 6 1 7 4 8 3) (6 3 7 2 8 5 1 4) (2 7 3 6 8 5 1 4)
 (7 3 1 6 8 5 2 4) (5 1 8 6 3 7 2 4) (1 5 8 6 3 7 2 4)
 (3 6 8 1 5 7 2 4) (6 3 1 7 5 8 2 4) (7 5 3 1 6 8 2 4)
 (7 3 8 2 5 1 6 4) (5 3 1 7 2 8 6 4) (2 5 7 1 3 8 6 4)
 (3 6 2 5 8 1 7 4) (6 1 5 2 8 3 7 4) (8 3 1 6 2 5 7 4)
 (2 8 6 1 3 5 7 4) (5 7 2 6 3 1 8 4) (3 6 2 7 5 1 8 4)
 (6 2 7 1 3 5 8 4) (3 7 2 8 6 4 1 5) (6 3 7 2 4 8 1 5)
 (4 2 7 3 6 8 1 5) (7 1 3 8 6 4 2 5) (1 6 8 3 7 4 2 5)
 (3 8 4 7 1 6 2 5) (6 3 7 4 1 8 2 5) (7 4 2 8 6 1 3 5)
 (4 6 8 2 7 1 3 5) (2 6 1 7 4 8 3 5) (2 4 6 8 3 1 7 5)
 (3 6 8 2 4 1 7 5) (6 3 1 8 4 2 7 5) (8 4 1 3 6 2 7 5)
 (4 8 1 3 6 2 7 5) (2 6 8 3 1 4 7 5) (7 2 6 3 1 4 8 5)
 (3 6 2 7 1 4 8 5) (4 7 3 8 2 5 1 6) (4 8 5 3 1 7 2 6)
 (3 5 8 4 1 7 2 6) (4 2 8 5 7 1 3 6) (5 7 2 4 8 1 3 6)
 (7 4 2 5 8 1 3 6) (8 2 4 1 7 5 3 6) (7 2 4 1 8 5 3 6)
 (5 1 8 4 2 7 3 6) (4 1 5 8 2 7 3 6) (5 2 8 1 4 7 3 6)
 (3 7 2 8 5 1 4 6) (3 1 7 5 8 2 4 6) (8 2 5 3 1 7 4 6)
 (3 5 2 8 1 7 4 6) (3 5 7 1 4 2 8 6) (5 2 4 6 8 3 1 7)
 (6 3 5 8 1 4 2 7) (5 8 4 1 3 6 2 7) (4 2 5 8 6 1 3 7)
 (4 6 1 5 2 8 3 7) (6 3 1 8 5 2 4 7) (5 3 1 6 8 2 4 7)
 (4 2 8 6 1 3 5 7) (6 3 5 7 1 4 2 8) (6 4 7 1 3 5 2 8)
 (4 7 5 2 6 1 3 8) (5 7 2 6 3 1 4 8))
```

Pegando como exemplo o último resultado da saída é: (5 7 2 6 3 1 4 8) e representando graficamente o tabuleiro 8x8 para melhor análise teríamos hipoteticamente assim a seguinte representação visual a seguir:

	1	2	3	4	5	6	7	8
1						L		
2			L					
3					L			
4							L	
5	L							
6				L				
7		L						
8								L

As posições das 8 rainhas estão indicadas por L no tabuleiro 8x8 e no resultado apresentado (5 7 2 6 3 1 4 8) a ordem de sequência indica a coluna e os números indicam as linhas do tabuleiro 8x8. Assim a primeira rainha se encontra na posição (5,1) onde 1 indica coluna e 5 linha, a segunda rainha se encontra na posição (7,2) onde 2 indica coluna e 7 linha, e assim sucessivamente.

#### 4.5. Solução da MDC para PP com a Linguagem C

O Problema do MDC entre dois números naturais no PP com a utilização da linguagem C foi bem mais fácil e com poucas linhas de código. Além da função principal main() foi preciso apenas criar uma função mdc() para realizar as operações necessárias a fim de obter o valor do MDC.

Esta função se baseia no algoritmo de Euclides que consiste em efetuar divisões sucessivas entre dois números até obter o resto zero. O máximo divisor comum entre os dois números iniciais é o último resto diferente de zero obtido. Inicialmente a função mdc() realiza a troca de variáveis caso seja necessário, porém a obtenção da resposta é dada por Euclides através do laço de repetição. A mdc() recebe os dois números como argumento e retorna o valor inteiro do MDC encontrado. A seguir apresentamos o código do algoritmo da função mdc():

```
int mdc(int a, int b){
    if (a != b) {
        if (b > a) {
            int aux = b;
            b = a;
            a = aux;
        }
        while (b != 0) {
            int r = a % b;
            a = b;
            b = r;
        }
    }
    return a;
}
```

A main() realiza a chamada a mdc() para que esta realize o cálculo e retorne o valor do mdc encontrado. Aqui foi colocado um laço de controle apenas para facilitar a execução e teste do mesmo, permitindo que o programa seja encerrado apenas caso

alguma letra diferente de “s” seja escolhida. A seguir é apresentada a classe principal:

```
int main(void){
    int n1, n2;
    char sair = 's';
    printf("\n O Maximo Divisor Comum (MDC) de dois numeros.\n");

    while(sair == 's'){
        printf("\n Entre com o primeiro Numero: ");
        scanf(" %d", &n1);
        printf(" Entre com o segundo Numero: ");
        scanf(" %d", &n2);
        printf(" O MDC ( %d , %d) = %d.\n",n1, n2, mdc(n1, n2));
        printf(" Se deseja continuar tecle s ou para sair qualquer
                                                    tecla: ");

        scanf(" %c", &sair );
    }
    return 0;
}
```

#### 4.6. Solução da MDC para POO com a Linguagem Java

O Problema do MDC entre dois números naturais no POO com a utilização da linguagem java foi muito parecido com a implementação realizada anteriormente em C (seção 4.5), apresentando poucas linhas de código e além do main() foi também preciso criar apenas um método mdc() para realizar as operações necessárias e encontrar o valor do MDC via solução do algoritmo de Euclides. As operações internas foram as mesmas, onde inicialmente o método mdc() pode realizar a troca de variáveis caso seja necessário e a obtenção da solução é dada pelo laço de repetição. A mdc() recebe os dois números como argumento e retorna o valor inteiro do MDC encontrado.

Diferente de C que utiliza a biblioteca stdio.h, java precisou realizar a importação com o comando import java.util.Scanner para realizar a obtenção dos valores fornecidos pelo usuário através da criação do objeto “dado” que é do tipo scanner na classe principal. Por isso o método main() se diferencia um pouco da implementação em C. Abaixo segue todo o código da resolução em java:

```
import java.util.Scanner;
// O máximo divisor comum (MDC) corresponde ao produto dos
```



```

divisores comuns entre dois ou mais números inteiros.
public class App {
    // Algoritmo de Euclides iterativo
    private static int mdc(int a, int b) {
        if (a != b) {
            if (b > a) {
                int aux = b;
                b = a;
                a = aux;
            }
            while (b != 0) {
                int r = a % b;
                a = b;
                b = r;
            }
        }
        return a;
    }

    public static void main(String[] args) {
        char sair = 's';
        Scanner dado = new Scanner(System.in);
        System.out.println(" \n O Maximo Divisor Comum (MDC) de
dois
numeros.");
        while (sair == 's') {
            System.out.print(" Entre com o primeiro Numero: ");
            int n1 = dado.nextInt();
            System.out.print(" Entre com o segundo Numero: ");
            int n2 = dado.nextInt();
            System.out.println(" O MDC (" + n1 + " ," + n2 + ") =
"
                                + mdc(n1,
n2));
            System.out.print(" Se deseja continuar tecle s ou para
sair qualquer tecla:
");
            sair = dado.next().charAt(0);
        }
        dado.close();
    }
}

```

```
}
```

#### 4.7. Solução da MDC para PL com a Linguagem Prolog

A lógica utilizada para a resolução do problema do MDC no PL com a utilização do prolog foi fácil. Matematicamente para computar o máximo divisor comum de dois números inteiros positivos dados, sendo eles X e Y, seu máximo divisor comum D pode ser encontrado segundo os três casos distintos:

- (1) Se X e Y são iguais, então D é igual a X;

$\text{mdc}(X, X, X).$

- (2) Se  $X < Y$ , então D é igual ao mdc entre X e a diferença  $X - Y$ ;

$\text{mdc}(X, Y, D) :-$

$X < Y,$

$Y1 \text{ is } Y - X,$

$\text{mdc}(X, Y1, D).$

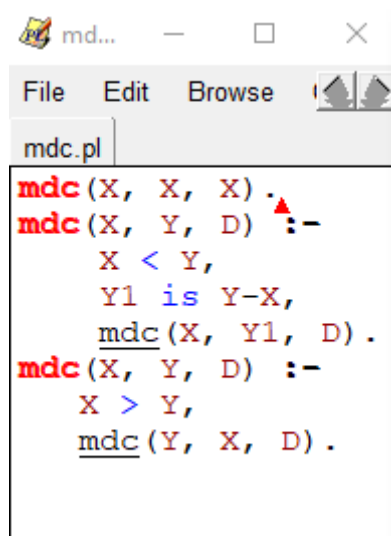
- (3) Se  $X > Y$ , então cai-se no mesmo caso (2), com X substituído por Y e vice-versa.

$\text{mdc}(X, Y, D) :-$

$X > Y,$

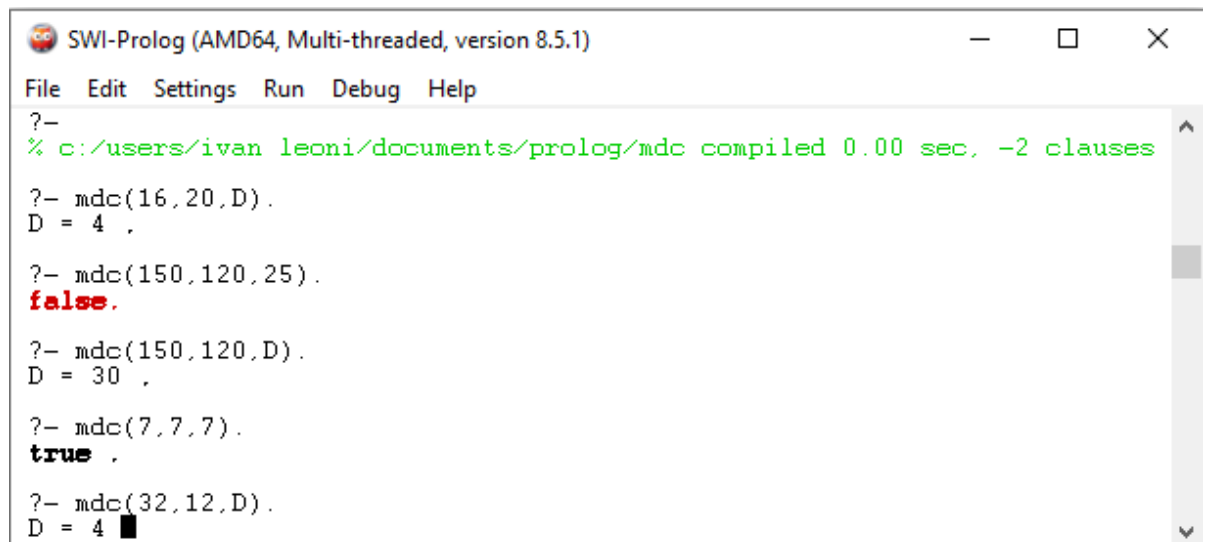
$\text{mdc}(Y, X, D).$

A seguir é apresentado o código do Prolog completo abrangendo os 3 casos:



```
mdc.pl
mdc(X, X, X) .
mdc(X, Y, D) :-
    X < Y,
    Y1 is Y-X,
    mdc(X, Y1, D) .
mdc(X, Y, D) :-
    X > Y,
    mdc(Y, X, D) .
```

A seguir é apresentado algumas possíveis soluções:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.5.1)
File Edit Settings Run Debug Help
?-
% c:/users/ivan leoni/documents/prolog/mdc compiled 0.00 sec, -2 clauses
?- mdc(16,20,D) .
D = 4 .
?- mdc(150,120,25) .
false.
?- mdc(150,120,D) .
D = 30 .
?- mdc(7,7,7) .
true .
?- mdc(32,12,D) .
D = 4
```

## 4.8. Solução da MDC para PF com a Linguagem Lisp

A lógica utilizada para a resolução do problema do MDC no PF com a utilização do common Lip é fácil e o código curto. Matematicamente o algoritmo de Euclides que diz que se  $r$  é o resto da divisão de  $a$  por  $b$ , então o maior divisor comum entre  $a$  e  $b$  é também o maior divisor comum entre  $b$  e  $r$ :  $\text{mdc}(a,b)=\text{mdc}(b,r)$ . Como é natural, quando o resto é zero, o maior divisor comum é o próprio  $b$ . Então em common lisp temos simplificadaamente as seguintes expressões:

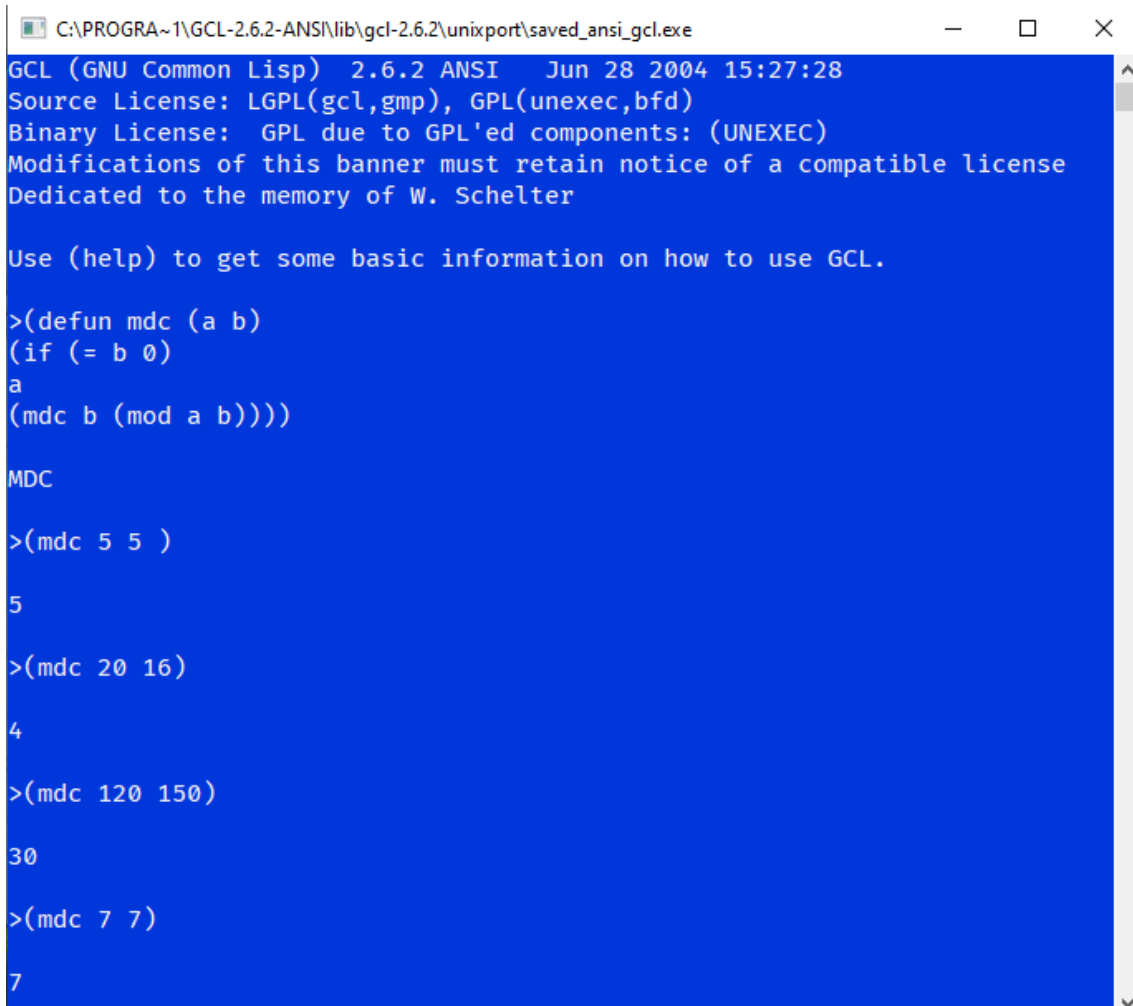
```
(defun mdc (a b)

  (if (= b 0)

    a
```

```
(mdc b (mod a b)))
```

A seguir é apresentado o código completo e alguns exemplos de solução em common Lisp:



```
C:\PROGRA~1\GCL-2.6.2-ANSI\lib\gcl-2.6.2\unixport\saved_ansi_gcl.exe
GCL (GNU Common Lisp) 2.6.2 ANSI Jun 28 2004 15:27:28
Source License: LGPL(gcl,gmp), GPL(unexec,bfd)
Binary License: GPL due to GPL'ed components: (UNEXEC)
Modifications of this banner must retain notice of a compatible license
Dedicated to the memory of W. Schelter

Use (help) to get some basic information on how to use GCL.

>(defun mdc (a b)
  (if (= b 0)
      a
      (mdc b (mod a b))))

MDC

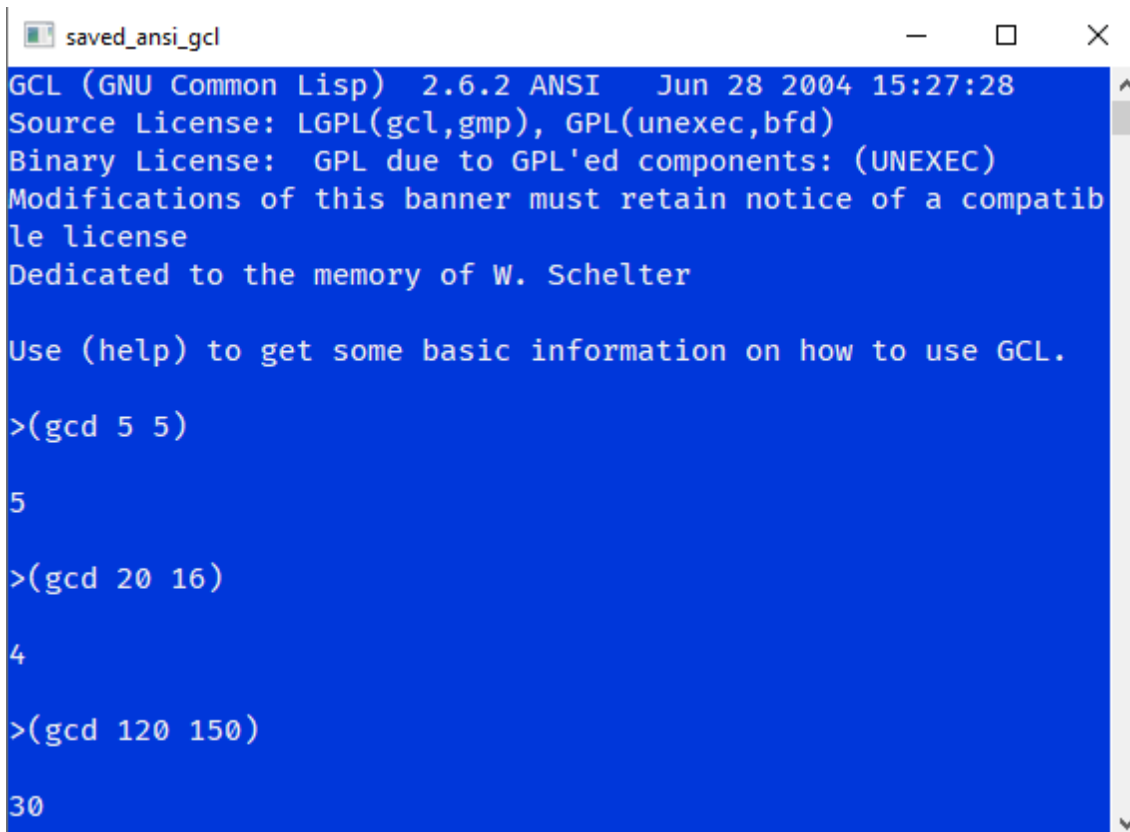
>(mdc 5 5 )
5

>(mdc 20 16)
4

>(mdc 120 150)
30

>(mdc 7 7)
7
```

No entanto Common Lisp apresenta uma função própria para determinar o mmc de dois ou mais números conforme pode ser observado a seguir:



```
saved_ansi_gcl
GCL (GNU Common Lisp) 2.6.2 ANSI Jun 28 2004 15:27:28
Source License: LGPL(gcl,gmp), GPL(unexec,bfd)
Binary License: GPL due to GPL'ed components: (UNEXEC)
Modifications of this banner must retain notice of a compatible license
Dedicated to the memory of W. Schelter

Use (help) to get some basic information on how to use GCL.

>(gcd 5 5)
5
>(gcd 20 16)
4
>(gcd 120 150)
30
```

#### 4.9. Solução da MMC para PP com a Linguagem C

O Problema do MMC para calcular o mínimo múltiplo comum entre dois números naturais na linguagem C é obtido utilizando duas funções além da principal. São elas: `mdc()` e `mmc()`, onde a função `mdc()` que auxilia a `mmc()` determinar o MMC já foi vista nas seções anteriores 4.5 e 4.6. Foi utilizada, então, a mesma estrutura já vista para a função `mdc()`. Conforme pode ser observado a seguir:

```
//maximo divisor comum
int mdc(int a, int b){
    if (a != b) {
        if (b > a) {
            int aux = b;
            b = a;
            a = aux;
        }
        while (b != 0) {
            int r = a % b;
            a = b;
```

```

        b = r;
    }
}
return a;
}

```

Para obter o valor do MMC foi utilizada a equação  $\text{mmc}(a,b) = a * b / \text{mdc}(a, b)$ . A função recebe os dois números como parâmetro de entrada e torna o valor do mmc, onde realiza uma chamada a função auxiliar `mdc()` para obter o valor e realizar os cálculos matemáticos básicos necessários para a solução. A seguir é apresentado o código utilizado na implementação do da função `mmc()`:

```

//minimo divisor comum
int mmc(int a, int b) {
    return a * b / mdc(a, b);
}

```

A seguir apresentamos a função `main()` utilizada para obter os dois números dos usuários e passá-los para a função `mmc` realizar os cálculos e retorná-los a `main()`, imprimindo a resposta ao usuário. O laço *while* de controle foi utilizado apenas para facilitar a execução e teste do mesmo, permitindo que o programa seja encerrado apenas caso alguma letra diferente de “s” seja escolhida.

```

int main(void) {
    int n1, n2;
    char sair = 's';
    printf("\n O minimo multiplo comum (MMC) de dois numeros.\n");
    while(sair == 's') {
        printf("\n Entre com o primeiro Numero: ");
        scanf(" %d", &n1);
        printf(" Entre com o segundo Numero: ");
        scanf(" %d", &n2);
        printf(" O MMC (%d , %d ) = %d.\n", n1, n2, mmc(n1, n2));
        printf(" Se deseja continuar tecle s ou para sair qualquer tecla: ");
        scanf(" %c", &sair);
    }
    return 0;
}

```

#### 4.10. Solução da MMC para POO com a Linguagem Java

O Problema do MMC para calcular o mínimo múltiplo comum entre dois números naturais no POO com a utilização da linguagem java foi muito parecido com a implementação realizada anteriormente em C (seção 4.9), apresentando poucas linhas de código e além do main() foi também preciso criar dois métodos: o mdc() e mmc(). Onde método mdc(), já visto anteriormente, é auxiliar do método mmc() e realiza as operações necessárias para encontrar o valor do MDC via solução do algoritmo de Euclides. A função mmc() chega a solução via chamada a mdc() e alguns cálculos matemáticos básicos.

Diferente de C que utiliza a biblioteca stdio.h, java precisou realizar a importação com o comando `import java.util.Scanner` para realizar a obtenção do valores fornecidos pelo usuário através da criação do objeto “dado” que é do tipo scanner na classe principal. Por isso, o método main() se diferencia um pouco da implementação em C. Abaixo segue todo o código da resolução em Java:

```
import java.util.Scanner;
public class App {
    // Algoritmo de Euclides iterativo
    private static int mdc(int a, int b) {
        if (a != b) {
            if (b > a) {
                int aux = b;
                b = a;
                a = aux;
            }
            while (b != 0) {
                int r = a % b;
                a = b;
                b = r;
            }
        }
        return a;
    }

    // Algoritmo do MMC
    private static int mmc(int a, int b) {
        return a * (b / mdc(a, b));
    }

    public static void main(String[] args) {
        char sair = 's';
        Scanner dado = new Scanner(System.in);
```

```

        System.out.println(" \nO minimo multiplo comum (MMC) de dois
numeros.");

        while (sair == 's') {
            System.out.print(" Entre com o primeiro Numero: ");
            int n1 = dado.nextInt();
            System.out.print(" Entre com o segundo Numero: ");
            int n2 = dado.nextInt();
            System.out.println(" O MMC (" + n1 + " ," + n2 + ") = " +
mmc(n1, n2));

            System.out.print(" Se deseja continuar tecle s ou para sair
qualquer tecla: ");
            sair = dado.next().charAt(0);
        }
        dado.close();
    }
}

```

#### 4.11. Solução da MMC para PL com a Linguagem Prolog

A lógica utilizada para a resolução do problema do MMC no PL com a utilização do prolog foi fácil e de código curto. Matematicamente temos que o mínimo múltiplo comum (mmc) de dois inteiros X e Y é o menor inteiro positivo que é múltiplo simultaneamente de X e de Y. Para computar o MMC de dois inteiros positivos dados, X e Y, seu MMC = Z pode ser encontrado segundo dois casos distintos:

- (1) Se X e Y são iguais, então Z é igual a X;

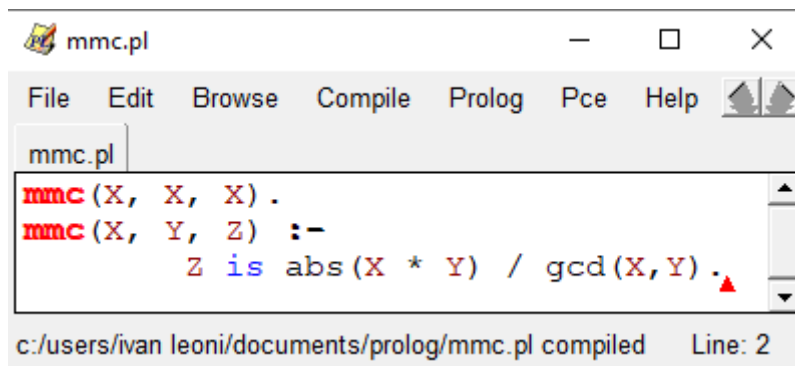
mmc(X, X, X).

- (2) Se  $X \neq Y$  então deve utilizar o máximo divisor comum (MDC) para obter o valor de Z. O SWI-Prolog reconhece a função **gcd** que determina o MDC. Assim o MMC pode ser obtido com a iteração de todos os múltiplos de X, até encontrar um que também seja múltiplo de Y. Resultado assim na seguinte fórmula:

$Z = |X * Y| / \text{MDC}(X, Y)$  ou em Prolog:  $Z \text{ is } \text{abs}(X * Y) / \text{gcd}(x, y)$ .

A seguir é apresentado o código completo para a resolução do MMC na linguagem Prolog:

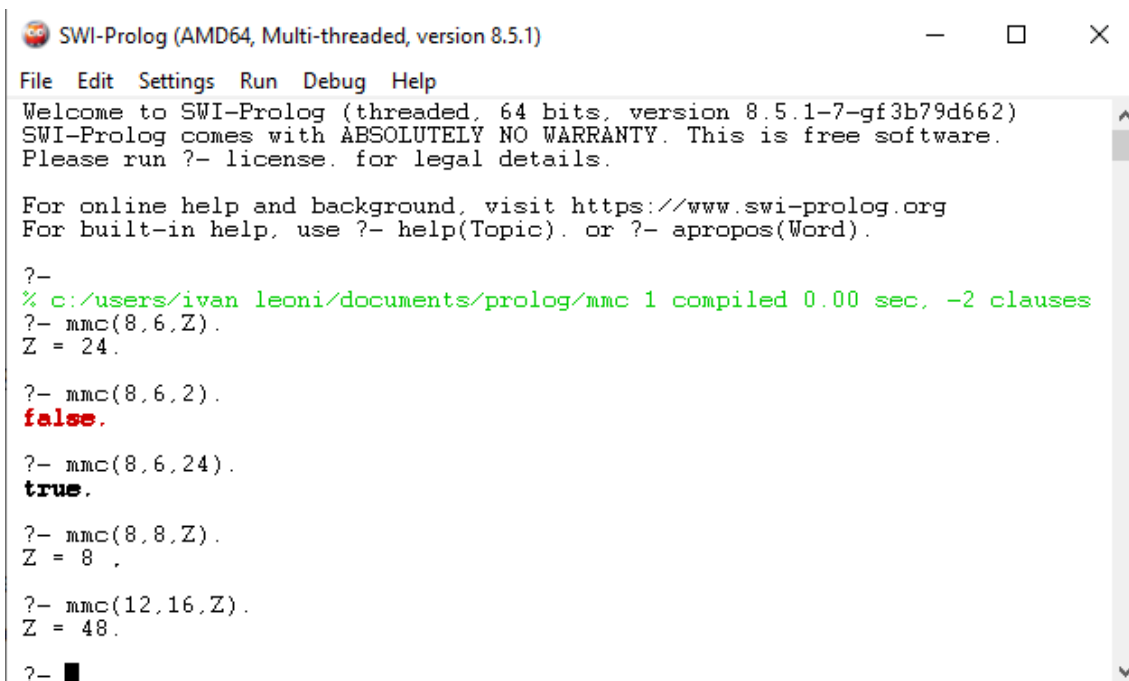




```
mmc(X, X, X) .  
mmc(X, Y, Z) :-  
    Z is abs(X * Y) / gcd(X, Y) .
```

c:/users/ivan leoni/documents/prolog/mmc.pl compiled Line: 2

A seguir é apresentado algumas possíveis soluções do MMC via SWI-Prolog:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.5.1)  
File Edit Settings Run Debug Help  
Welcome to SWI-Prolog (threaded, 64 bits, version 8.5.1-7-gf3b79d662)  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  
Please run ?- license. for legal details.  
  
For online help and background, visit https://www.swi-prolog.org  
For built-in help, use ?- help(Topic). or ?- apropos(Word).  
  
?-  
% c:/users/ivan leoni/documents/prolog/mmc 1 compiled 0.00 sec, -2 clauses  
?- mmc(8,6,Z).  
Z = 24.  
  
?- mmc(8,6,2).  
false.  
  
?- mmc(8,6,24).  
true.  
  
?- mmc(8,8,Z).  
Z = 8.  
  
?- mmc(12,16,Z).  
Z = 48.  
  
?-
```

#### 4.12. Solução da MMC para PF com a Linguagem Lisp

A resolução do problema do MDC no PF em common Lip apresenta um curto código, a solução lógica se baseia matematicamente em que o mínimo múltiplo comum (mmc) de dois inteiros m e n é o menor inteiro positivo que é múltiplo simultaneamente de m e de n. O Common Lisp também reconhece a função gcd (MDC) que auxilia a encontrar o valor do MMC entre dois números, ao dividir o resultado do gcd pelo valor da multiplicação de m x n. Caso seja fornecido valor para m ou n sendo iguais aos valor zero será então retornado zero, pois todo número é múltiplo de zero.

A seguir é apresentado o código completo para a solução do MMC de dois inteiros na

linguagem Common Lisp:

```
(defun mmc (&rest args)

  (reduce (lambda (m n)

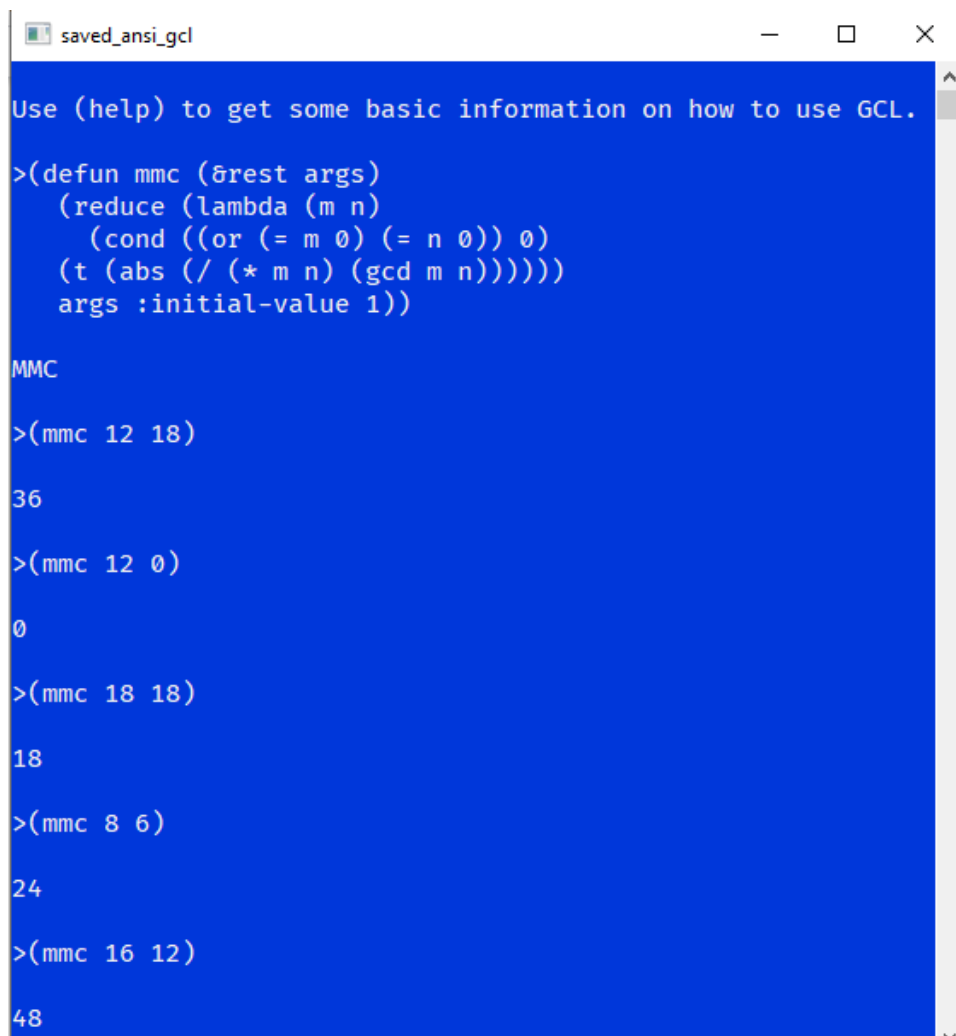
    (cond ((or (= m 0) (= n 0)) 0)

    (t (abs (/ (* m n) (gcd m n))))))

  args :initial-value 1))
```

No código acima o **lambda** encontra o MMC de dois inteiros. A redução o transforma para aceitar qualquer quantidade de números inteiros. A operação de redução explora como mmc é associativo,  $(\text{lcm } a \ b \ c) == (\text{lcm } (\text{lcm } a \ b) \ c)$ ; e como 1 é uma identidade,  $(\text{mmc } 1 \ a) == a$ .

A seguir são apresentados alguns possíveis exemplos de soluções.



```
saved_ansi_gcl

Use (help) to get some basic information on how to use GCL.

>(defun mmc (&rest args)
  (reduce (lambda (m n)
    (cond ((or (= m 0) (= n 0)) 0)
    (t (abs (/ (* m n) (gcd m n))))))
  args :initial-value 1))

MMC

>(mmc 12 18)
36

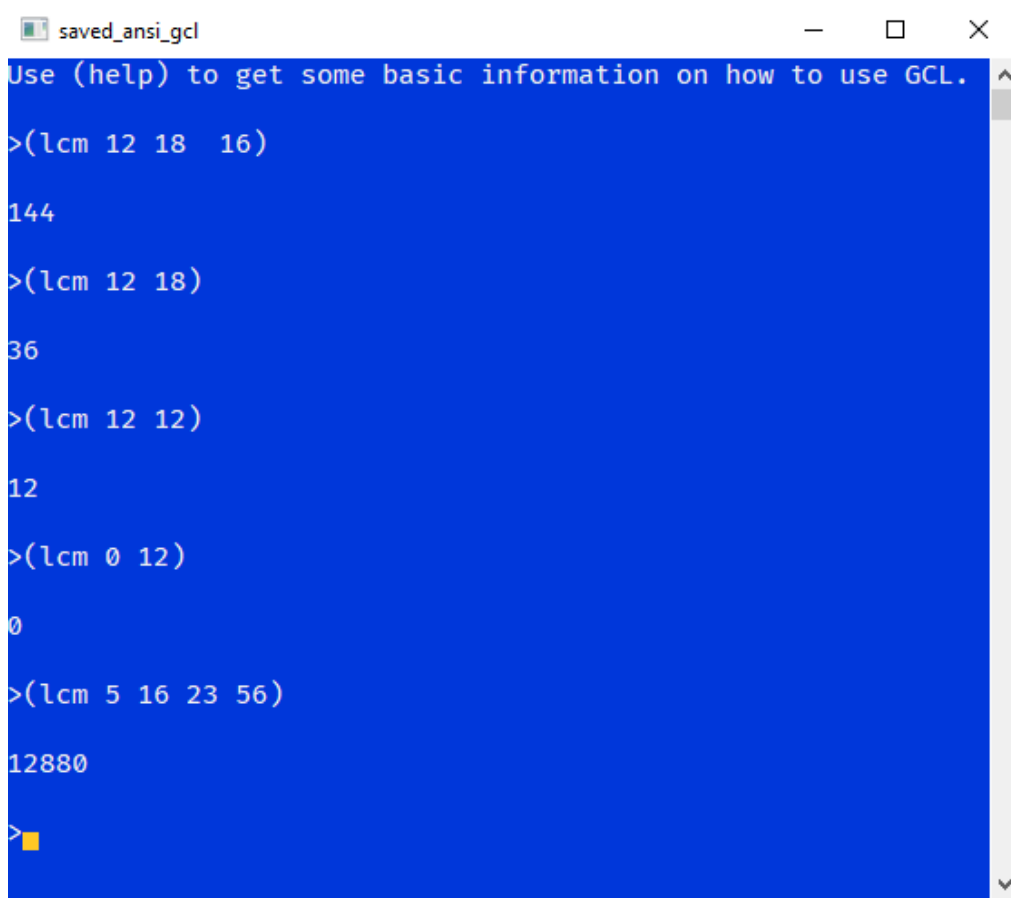
>(mmc 12 0)
0

>(mmc 18 18)
18

>(mmc 8 6)
24

>(mmc 16 12)
48
```

No entanto Common Lisp apresenta uma função própria para determinar o mmc de dois ou mais números conforme pode ser observado a seguir:



```
Use (help) to get some basic information on how to use GCL.

>(lcm 12 18 16)

144

>(lcm 12 18)

36

>(lcm 12 12)

12

>(lcm 0 12)

0

>(lcm 5 16 23 56)

12880

>
```

## 5. Análise dos Paradigmas e Linguagens

As principais características observadas neste projeto com relação a todos os paradigmas e as linguagens utilizadas para a realização dos problemas no quesito implementação, tamanho de código e legibilidade são apresentados na tabela 02.

A tabela 02 foi realizada conforme as observações pessoais dos integrantes deste projetos, ou seja, conforme a experiência da equipe ao realizar as soluções dos problemas, certamente a experiência e prévio contato com a linguagem e o paradigma influenciou diretamente na categorização das características apresentadas na Tabela 02. Ou seja, a mesma poderá ser retratada diferentemente por quem já teve ou tem um contato maior com tais linguagens. A seguir apresentamos mais detalhadamente os resultados da tabela 02.

Problema	Características	Linguagens e Paradigmas			
		C (PP)	Java(POO)	Prolog (PL)	Lisp(PF)
8R	Lógica da Solução	Fácil	Fácil	Difícil	Difícil
MDC		Fácil	Fácil	Fácil	Fácil
MMC		Fácil	Fácil	Fácil	Fácil
8R	Implementação	Fácil	Fácil	Difícil	Difícil
MDC		Fácil	Fácil	Fácil	Fácil
MMC		Fácil	Fácil	Fácil	Fácil
8R	Tamanho do Código	Grande	Grande	Grande*	Curto
MDC		Grande	Grande	Curto	Curto
MMC		Grande	Grande	Curto	Curto
8R	Legibilidade	Fácil	Fácil	Difícil	Difícil
MDC		Fácil	Fácil	Fácil	Fácil
MMC		Fácil	Fácil	Fácil	Médio

*Tabela 02: Comparação entre as linguagens utilizadas*

A linguagem que foi mais fácil de implementar as soluções foi a linguagem C e Java, porém Java foi ainda mais fácil, uma vez que, o PP com linguagem C contribuiu para a solução dos problemas em POO, pois como ambos são do Paradigma imperativos não houve a necessidade de mudar a forma da resolução de C para Java, a lógica de solução foi a mesma assim o POO teve previamente uma base bem estabelecida na resolução em C. E por fim, levando em conta que apesar da lógica de solução dos problemas do mmc e mdc ser fácil de entender, o problema das 8 rainhas foi de difícil solução lógica e de implementação nas linguagem prolog e lisp.

Com relação ao tamanho do código, a linguagem que apresentou menor linhas de código foi a linguagem Lisp e Prolog, em comparação a extensa camada de código

quando se trata da linguagem C e Java. Vale ressaltar que o das 8 rainhas em Prolog teve um tamanho grande, mas somente devido a parte da solução gráfica, porém, o mesmo poderia ter sido solucionado sem a parte gráfica. Apresentando assim um código bem menor se desconsideramos a parte gráfica.

Já com relação a legibilidade C e Java foi muito mais fácil de entender a codificação quando comparadas as linguagens Prolog e Lisp quanto ao problema das 8 rainhas. Isso se deve também ao fato do desconhecimento dos integrantes da equipe sobre essas linguagens. Ao analisarmos os códigos, observamos que quando usamos C e Java obtemos praticamente o mesmo código. Isso se dá, também, por ambas se tratarem do paradigma imperativo. As diferenças se limitam a algumas pequenas funcionalidades que uma linguagem tem e outra não tem e vice-versa. Contudo, se teve mais facilidade quanto ao desenvolvimento em C e Java por serem linguagens de conhecimento prévio já estabelecidas dos realizadores deste trabalho.

Verificou-se que quanto maior a complexidade do problema matemático a ser resolvido mais a complexidade de legibilidade do código em Common Lisp vai se tornando também, por isso o mmc do lisp classificamos como médio, pois aumentou-se o código e a dificuldade do seu entendimento quanto comparado ao mdc na mesma linguagem.

Por fim, além das linguagens LISP e PROLOG terem sido um pouco mais difíceis de serem desenvolvidas por nunca terem sido utilizadas anteriormente pelos realizadores deste projeto, constatou-se que estas duas linguagens em relação a quantidade de código (linhas) usadas foi incrivelmente menor que C e Java. Isso se dá ao fato de LISP e Prolog utilizarem a recursividade para resolver os problemas. Além disso, estas duas linguagens são melhores empregadas para resoluções de problemas matemáticos como os dois últimos: mmc e mdc.

## 6. Conclusão

Para este projeto todos os problemas puderam ser implementados em todos os 4 paradigmas e nas respectivas 4 linguagens propostas, sem exceção.

Todos os problemas apresentaram a resolução em Prolog e Common Lisp com um código mais limpo e menor. Contudo o desafio para implementar, levando em conta a falta de prática destas linguagens, foi superior comparado a C e Java, principalmente no problema das 8 rainhas.

Referente ao problema específico das 8 rainhas devido a apresentação visual da solução e levando em conta um maior entendimento do código, da prática das linguagens e sobre os prévios conhecimentos da POO e PP indicamos o uso da linguagem C ou Java para o desenvolvimento da solução.

Já como os dois últimos problemas são puramente matemáticos, o PF ou PL certamente são ambos indicados, uma vez que o PF se baseia nas funções matemáticas e o PL utiliza lógica matemática para a resolução dos mesmos e ainda considerando que o Common Lisp e SWI-Prolog já apresentam até mesmo estas funções específicas que determinam o MMC e MDC, não precisando assim codificá-las. E por fim, mesmo quando codificando as soluções do MMC e MDC seja em common Lisp ou em Prolog, a nível de aprendizado, elas apresentaram de uma forma geral, quando comparada a C e Java, um código bem mais curto e uma lógica de resolução relativamente fácil.

## Referências:

- The C Language. Disponível em: [www.cs.utah.edu](http://www.cs.utah.edu). Acesso em 06/11/2021.
- Java™ Programming Language. Disponível em: <https://docs.oracle.com/>. Acesso em 06/11/2021.
- Lecture Notes, An Introduction to Prolog Programming, Ulle Endriss. Disponível em: <https://ocw.upj.ac.id/files/Textbook-TIF212-Prolog-Tutorial-3.pdf>. Acesso em 06/11/2021.
- Common Lisp. Disponível em <https://lisp-lang.org/>. Acesso em 06/11/2021.
- Ana Cristina Vieira de Melo e Flávio Soares Corrêa da Silva. Princípios de Linguagens de Programação. 3a edição. Blucher. 2003.
- Colmerauer, A., & Roussel, P. (1996). The birth of Prolog. In History of programming languages---II (pp. 331-367).
- Vinícius, T. Java: história e principais conceitos. 2012. Disponível em: <<https://www.devmedia.com.br/java-historia-e-principais-conceitos/25178>>. Acesso em 06/11/2021.
- Lisp. Disponível em: <<https://programacao.fandom.com/wiki/Lisp>>. Acesso em 06/11/2021.
- SANTANA, M. Linguagens Imperativas e linguagens funcionais. Disponível em: [http://www.ppgsc.ufrr.br/~rogerio/material\\_auxiliar/CLP20131\\_linguagens\\_imperativas\\_funcionais.pdf](http://www.ppgsc.ufrr.br/~rogerio/material_auxiliar/CLP20131_linguagens_imperativas_funcionais.pdf) . Acesso em 06/11/2021.
- CASAVELLA, E. Breve história da linguagem C. Disponível em: <http://linguagemc.com.br/breve-historia-da-linguagem-c/>. Acesso em 06/11/2021.
- MAURO, J. Paradigma Funcional. Disponível em: <http://informacaocomdiversao.blogspot.com.br/2009/02/paradigma-funcional.html> Acesso em 06/11/2021.
- GUSTAVO. Paradigmas funcionais: Funções matemáticas e funções em linguagens de programação. Disponível em: <https://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula15.pdf> Acesso em 07/11/2021.
- EDERSON, C. et al. Programação em lógica. Disponível em: <https://hudsoncosta.files.wordpress.com/2011/05/programacaoemlogica.pdf> Acesso em 07/11/2021

Henrique, 2014. Os 4 pilares da programação orientada a objetos. Disponível em: <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264> acesso em 07/11/2021 às 14h30.

Manssour, I. H. Paradigma Orientado a objeto. Disponível em: <https://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula12.pdf> acesso em 07/11/2021 às 15h00.

Simões, E; Faria, L; Ferraz, M; Gondim, R; Rafael, R.. “LISP Linguagem de Programação”. UFG. Disponível em: <http://www.inf.ufg.br/~eduardo/lp/alunos/lisp/historico.html>. Acesso em 07/11/2021 às 16h00.

Eduardo. “Prolog”. UFG. Disponível em: <http://www.inf.ufg.br/~eduardo/lp/alunos/prolog/prolog.html>. Acesso em 07/11/2021.

Santos, “Trabalho 01”. USP. Disponível em: <https://sites.icmc.usp.br/sandra/18/trabalho01.html>. Acesso em 07/11/2021.

Compilador Lisp Online. Disponível em: [http://rextester.com/l/common\\_lisp\\_online\\_compiler](http://rextester.com/l/common_lisp_online_compiler) . Acesso em 07/11/2018.

Compilador Prolog Online. Disponível em: <https://swish.swi-prolog.org>. Acesso em 07/11/2018.

Glossário Prolog. Disponível em: <http://kodumaro.blogspot.com/2014/05/pequeno-glossario-de-prolog.html>. Acesso 10/11/2021.

Linguagem LISP. Disponível em: <https://www.dca.ufrn.br/~adelardo/lisp/>. Acesso em 10/11/2021.

Máximo Divisor Comum - LISP. Disponível em: <https://www.ic.unicamp.br/~meidanis/courses/mc336/problemas-lisp/p32.lisp>. Acesso em 10/11/2021.

Problema Oito Rainhas. Disponível em: [http://www.vision.ime.usp.br/~pmiranda/mac122\\_2s14/aulas/aula20/aula20.html](http://www.vision.ime.usp.br/~pmiranda/mac122_2s14/aulas/aula20/aula20.html). Acesso em 11/11/2021.