CLIPS

INSTITUTO FEDERAL DE CIÊNCIAS E TECNOLOGIA DE SÃO PAULO, CÂMPUS CUBATÃO

CLIPS

AUTORES

Estela Almeida Alves
Kettlyn Gabrielly Lima Marcelino
Letícia de Araújo Machado
Lucas de Souza Vieira
Mariana Almeida Alves
Maria Vitorya de Souza Silva
Matheus Henrique Valdivino Costa
Nathally Santos Barros
Stephany da Costa Silva

ORIENTADOR
Mauricio Neves Asenjo

Índice

1. CAPÍTULO 1

1.1 INTELIGÊNCIA ARTIFICIAL

1.1.1 O QUE É?

Inteligência artificial (IA) refere-se a qualquer comportamento semelhante ao do humano desenvolvido por uma máquina ou sistema. Na forma mais básica da IA, os computadores são programados para "imitar" o comportamento humano usando dados extensivos de exemplos anteriores de comportamento similar. Eles podem variar desde identificar comportamentos básicos até realizar atividades complexas em uma fábrica.

Embora a IA traga imagens de robôs parecidos com os homens de alto funcionamento que dominam o mundo, a IA não pretende substituir os seres humanos. Seu objetivo é melhorar significativamente as habilidades e contribuições humanas. Isso faz dela um ativo de negócios muito valioso.

1.1.2 COMO SURGIU A INTELIGÊNCIA ARTIFICIAL?

Antes de 1949, os computadores executavam os comandos, mas não conseguiam se lembrar do que haviam feito, porque não armazenavam esses comandos. Em 1950, Alan Turing discutiu como criar máquinas inteligentes e testar essa inteligência em seu estudo "Máquinas computacionais e inteligência". Cinco anos depois, o primeiro projeto de IA foi apresentado no Projeto de Pesquisa de Verão do Dartmouth sobre inteligência artificial (DSPRAI).

Entre 1957 e 1974, os computadores ficaram mais rápidos, baratos e acessíveis. Os algoritmos de Machine Learning melhoraram e, em 1970, um dos idealizadores do DSPRAI disse à revista Life que haveria daqui três a oito anos uma máquina com a inteligência geral de um ser humano médio. Apesar de seu sucesso, os computadores ainda não conseguem armazenar e processar informações de forma rápida e eficiente, criando obstáculos para a busca por inteligência artificial na próxima década.

A Inteligência Artificial retornou na década de 1980 com a evolução do kit de ferramentas de algoritmos e fundos mais dedicados. John Hopefield e David Rumelhart apresentaram as técnicas de “deep learning” que permitiram que os computadores aprendessem com experiência. Edward Feigenbaum apresentou os “sistemas especialistas” que simulavam a tomada de decisões humana. Apesar da falta de financiamento governamental e publicidade, a IA prosperou e muitos marcos importantes foram alcançados nas duas décadas seguintes. Em 1997, o então atual campeão mundial e grande mestre de xadrez Gary Kasparov foi derrotado pelo Deep Blue da IBM, um programa de computador que joga xadrez com humanos. No mesmo ano, o software de reconhecimento de fala desenvolvido pela Dragon Systems foi implementado no Windows. Cynthia Breazeal também desenvolveu o Kismet, um robô que conseguia reconhecer e demonstrar emoções.

Em 2016, o programa AlphaGo do Google derrotou o mestre Lee Se-dol em Go e, em 2017, Libratus, um supercomputador jogador de pôquer derrotou os melhores jogadores humanos, está é um pouco da história da IA e sua evolução ao longo dos anos.

1.1.3 TECNOLOGIAS POR TRÁS

Dentro do campo da Inteligência Artificial existem tecnologias que contribuem para que ela evolua. Veja a seguir as principais:

Machine Learning – É o processo pelo qual os computadores desenvolvem o reconhecimento de padrões ou a capacidade de aprender continuamente ou fazer previsões com base em dados, e podem fazer ajustes sem serem especificamente programados para isso. Uma forma de inteligência artificial, o Machine Learning automatiza com eficiência o processo de construção de modelos analíticos e permite que as máquinas se adaptem a novos cenários de forma independente.

Plataformas de Machine Learning são capazes de fornecer capacidade computacional, bem como dados, algoritmos, APIs, entre outras soluções para se projetar, treinar e aplicar modelos da área em máquinas, aplicativos, processos etc.

Deep Learning – É um subconjunto do Machine Learning que tem um desempenho significativamente superior ao de alguns métodos tradicionais do Machine Learning. Utiliza-se de uma combinação de redes neurais artificiais de várias camadas com uso intenso de dados e de computação, inspirados na compreensão do comportamento do cérebro humano. Essa abordagem tornou-se tão eficaz que até começou a ultrapassar as habilidades humanas em várias áreas, como reconhecimento de imagem e voz e processamento de linguagem natural.

Os modelos de Deep Learning processam grandes quantidades de dados e são geralmente não supervisionados ou semissupervisionados.

Processamento de Linguagem Natural (PLN) – Por meio do PLN, as máquinas podem compreender melhor os textos humanos, envolvendo o reconhecimento de contextos, extração de informações, desenvolvimento de resumos. Para isso é empregado softwares, programação e outras soluções, também é possível compor textos partindo de dados obtidos por computadores, o PLN pode ser usado em áreas como atendimento ao consumidor e na produção de relatórios corporativos.

1.1.4 TIPOS DE INTELIGÊNCIAS ARTIFICIAIS

Há diferentes tipos de Inteligência Artificial, cada um com seus próprios objetivos e abordagens. Entre as principais categorias, podemos dividi-las em:

·Inteligência de Máquina – Tem o objetivo de criar máquinas capazes de realizar tarefas específicas programadas para reconhecer padrões, tomar decisões lógicas ou aprender a partir de experiências anteriores. Sua aplicação tem como objetivo automatizar processos e simplificar tarefas diárias.

·Inteligência Humanoide – Visa desenvolver máquinas capazes de “imitar “o comportamento humano. Estás maquinas são programadas para se comportar de forma semelhante aos seres humanos, realizando tarefas como conversar, reconhecer faces ou expressar emoções. Sua aplicação tem a ver com a criação de interfaces digitais mais humanizadas e inteligentes em dispositivos.

·Inteligência Coletiva – Busca criar redes complexas capazes de tomar decisões coletivas. Essas redes são formadas por múltiplos agentes inteligentes interconectados, que trabalham juntos para solucionar problemas. O intuito é criar sistemas complexos capazes de tomar decisões coletivas, como um sistema financeiro inteligente.

Há diferentes tipos de IA no que diz respeito às características operacionais, incluindo as seguintes:

· **IA Reativa** – Funciona com base em regras pré-programadas e não é capaz de aprender com a experiência ou construir uma memória.

· **IA de Memória Limitada** – Pode se lembrar de experiências passadas e usá-las para informar as decisões futuras.

· **IA de Teoria da Mente** – É capaz de entender estados mentais tais como crenças, desejos e intenções.

· **IA Autoconsciente** – Está ciente de seus próprios estados mentais e pode usar esta informação para raciocinar sobre o mundo.

1.2 O CLIPS

O CLIPS (Sistema de Produção Integrada em Linguagem C, em inglês) é uma ferramenta de software de domínio público para construção de sistemas especialistas (SE) uma linguagem própria para essa finalidade. A sintaxe e o nome foram inspirados no OPS5 de Charles Forgy, que foi desenvolvido no final dos anos 70.

A partir de 1985, as primeiras versões do CLIPS foram desenvolvidas no centro Johnson Space Center da NASA e tinham como finalidade gerar soluções que apresentassem alta portabilidade, baixo custo e fácil integração com sistemas externos, mas esse processo foi interrompido, em meados de 1990, quando as responsabilidades do grupo de desenvolvimento deixaram de se concentrar na tecnologia de sistemas especialistas. O nome original do projeto era "NASA's AI Language" (NAIL).

O CLIPS em si é escrito em C, as suas extensões podem ser escritas em C e o próprio CLIPS pode ser chamado de C. A sua sintaxe é muito semelhante com a da linguagem de programação LISP. Ele incorpora uma linguagem completa orientada a objetos para escrever sistemas especialistas e usa encadeamento direto, ou seja, ele interpreta as informações para depois gerar um resultado. Assim como outras linguagens de sistemas especialistas, o CLIPS lida com regras e fatos. Vários fatos podem se tornar uma regra aplicável e depois uma regra aplicável pode ser acionada. Ele também é caracterizado por sua portabilidade, pode ser integrado com outras linguagens como C/C++ e Ada através de chamadas procedurais. Seu desempenho é garantido pelo algoritmo RETE utilizado no motor de inferência.

Como uma ferramenta de auxílio para as tomadas de decisões de um sistema especialista, já foi muito utilizada para o desenvolvimento de sistemas, porém, uma das preocupações dos desenvolvedores continua sendo a interface que o CLIPS fornece, que não permite muita interação com o usuário. Para essa questão, a solução encontrada foi desenvolver a interface de um SE utilizando uma outra linguagem de programação orientada a objetos (POO) que permitisse uma interface mais interativa.

1.2.1 QUE PODE SER DESENVOLVIDO NO CLIPS

O CLIPS é um aplicativo versátil, onde podem ser desenvolvidos diversos programas com funcionalidades totalmente distintas, suas principais construções definidoras são: defmodule, defrule, deffacts, deftemplate, defglobal, deffunction, defclass, definstances, defmessage-handler, defgeneric e método def.

Todas as construções citadas seguem o mesmo modelo: estão entre parênteses e com exceção da 'defglobal' permitem acrescentar diretamente um comentário. Os comentários também podem ser colocados no código CLIPS usando um ponto e vírgula (;). Tudo que for incluído do ponto e vírgula até o próximo caractere de retorno será ignorado pelo CLIPS. Se o ponto e vírgula for o primeiro caractere da linha, toda a linha será tratada como um comentário. Vale ressaltar que ao contrário das chamadas de função, as construções nunca têm um valor de retorno.

Existem três formatos principais para representar informações no CLIPS. São elas:

1.2.2 FACTS (FATOS)

Denominados como a unidade fundamental, os fatos podem ser incluídos a lista de fatos usando o comando 'assert', removidos usando o comando 'retract', modificados através do 'modify' ou duplicados usando o comando 'duplicate' por meio de interação explícita do usuário ou enquanto um programa CLIPS é executado. Se um fato for declarado e já corresponder a algum existente dentro da lista de fatos, a nova asserção será ignorada (no entanto, esse comportamento pode ser alterado usando a função set-fact-duplication).

Os comandos retract, modify e duplicate, exigem que um fato seja especificado, seja por índice de fato ou endereço de fato. O índice de fato é um parâmetro inteiro e único, denominado logo após a declaração do usuário; o valor padrão inicial é um, os demais são incrementados a partir do valor anterior somado com 1, por exemplo, a primeira denominação tem o índice de fato valendo 1, a segunda 2 (1+1), a terceira 3 (2+1), e assim por diante. Quando um fato é modificado, seu índice de fatos permanece inalterado. Sempre que um comando reset ou clear é dado, os índices de fatos são reiniciados em um.

O endereço de fato pode ser obtido capturando o valor de retorno de comandos que retornam endereços de fato (como assert, modify e duplicar) ou vinculando uma variável ao endereço de fato de um fato que corresponda a um padrão no LHS de uma regra.

Um identificador de fato é uma notação abreviada para exibir um fato. Consiste no caractere 'f', seguido de um traço, seguido do índice de fato do fato. Por exemplo, f-10 refere-se ao fato com índice de fato 10. Um fato é armazenado em um dos dois formatos: ordenado ou não ordenado.

1.2.3 ORDERED FACTS (FATOS ORDENADOS)

Consistem em um símbolo seguido por uma sequência de zero ou mais campos que são separados por espaços e delimitados por parênteses. O primeiro campo de um fato ordenado especifica uma "relação" que se aplica aos demais. Por exemplo, (pai de jack bill) afirma que bill é o pai de jack.

Exemplos de fatos ordenados:

(the pump is on) a bomba está ligada (tradução)

(altitude is 10000 feet) altitude é 10.000 pés (tradução)

(grocery-list bread milk eggs) ovos de leite de pão de lista de compras (tradução)

Nenhuma restrição é imposta na ordem dos campos, exceto no primeiro, que deve obrigatoriamente ser um símbolo. Entretanto, existem símbolos reservados que só podem ser usadas como um nome padrão (definido explicitamente ou implícito) e não devem ser usados como primeiro campo em nenhum fato (ordenado ou não ordenado), são eles: test, and, or, not, declare, logical, object, exists, e forall. Ademais, esses símbolos podem ser usados como nomes de slots, no entanto, não é recomendado.

1.2.4 NON-ORDERED FACTS (FATOS NÃO ORDENADOS)

Os fatos não ordenados (ou padrão) fornecem ao usuário a capacidade de abstrair a estrutura de um fato atribuindo nomes a cada campo. Para facilitar a busca através de nomes, é recomendado que utilize a construção deftemplate, por meio dela é possível criar um modelo que pode ser usado para acessar campos por nome, sendo assim, é considerada análoga a uma estrutura em C.

Vale ressaltar que a construção deftemplate permite que o nome de um template seja definido junto com zero ou mais definições de slots. Ao contrário dos fatos ordenados, os slots de um fato padrão podem ser restringidos por tipo, valor e intervalo numérico. Um slot consiste em um parêntese de abertura seguido pelo nome do slot, zero ou mais campos e um parêntese de fechamento. Além disso, os slots não podem ser usados em um fato ordenado e as informações em um fato padrão não podem ser referenciadas posicionalmente.

Os fatos Deftemplate e os fatos ordenados pelo primeiro campo são distintos. O primeiro campo de todos os fatos deve ser um símbolo, no entanto, se esse símbolo corresponder ao nome de um padrão, então o fato é um fato padrão. O primeiro campo de um fato padrão é seguido por uma lista de zero ou mais slots.

Exemplos de fatos não ordenados:

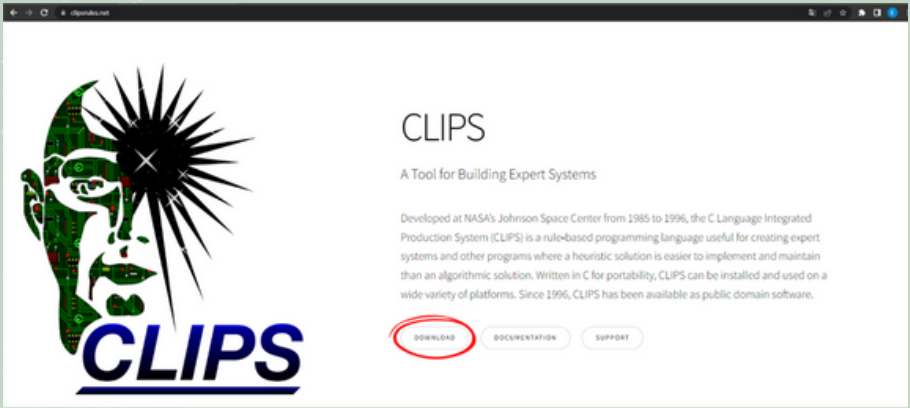
```
(client (name "Joe Brown") (id X9345A))
 (point-mass (x-velocity 100) (y-velocity -200))
  (class (teacher "Martha Jones") (#-students 30) (Room "37A"))
 (grocery-list (#-of-items 3) (items bread milk eggs))
```

1.2.5 INITIAL FACTS (FATOS INICIAIS)

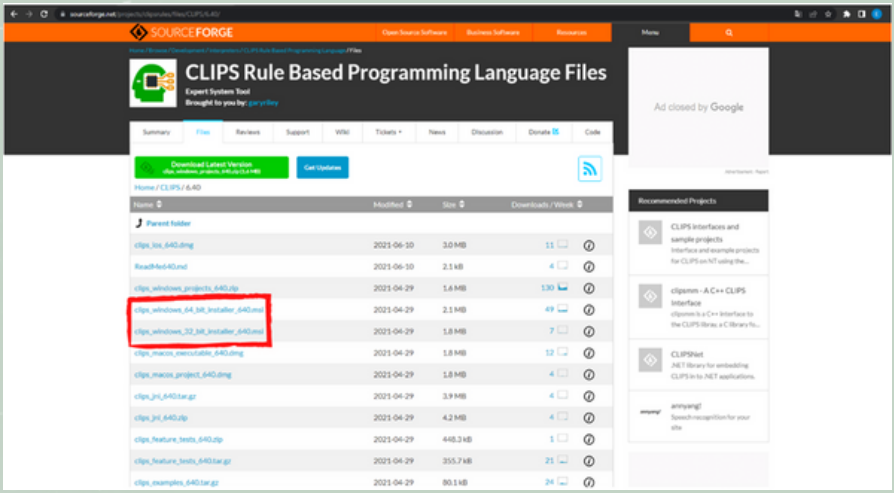
A construção deffacts permite que um conjunto de conhecimentos, como a priori ou inicial seja especificado como uma coleção de fatos. Quando o ambiente CLIPS é redefinido (por meio do comando reset), todos os fatos especificados em uma construção deffacts na base de conhecimento do CLIPS são adicionados à lista de fatos automaticamente.

1.3 PASSO A PASSO DA INSTALAÇÃO

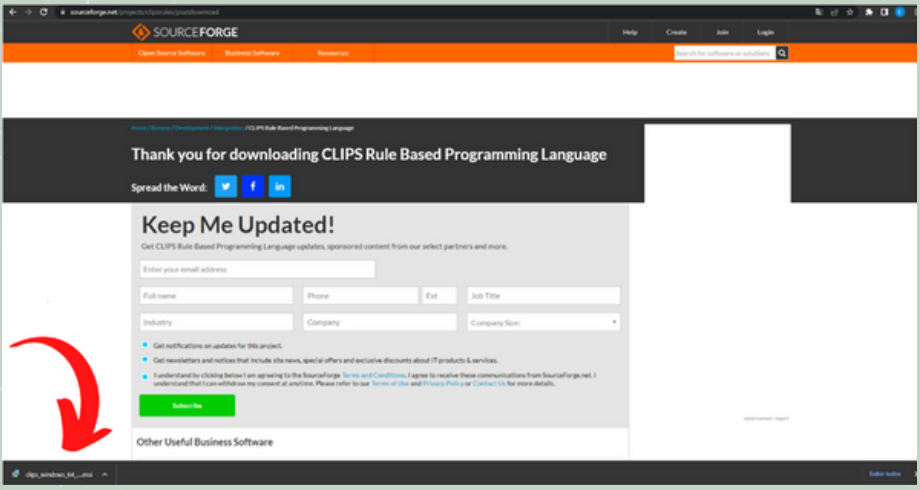
PASSO 1: Acessar o site https://www.clipsrules.net/. Para instalar a IDE em seu computador, é preciso clicar em download.



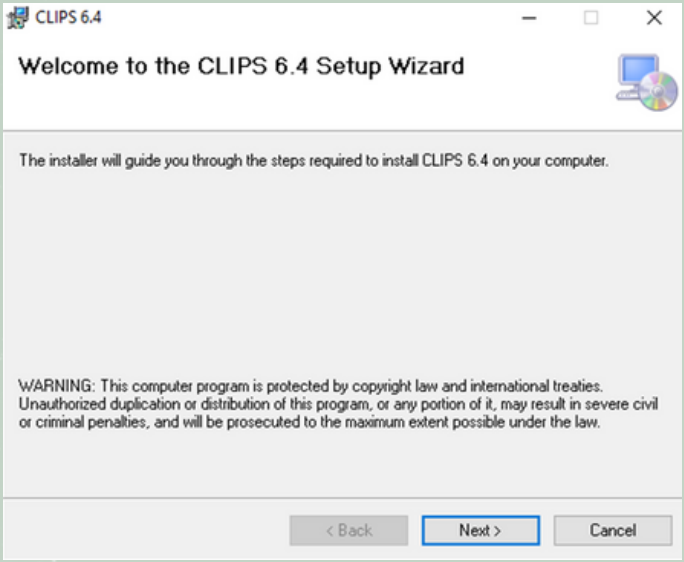
PASSO 2: Você será redirecionado para o site SourceForge. Entre as opções, selecione a opção de 32x ou 64x, de acordo com as configurações do seu computador.



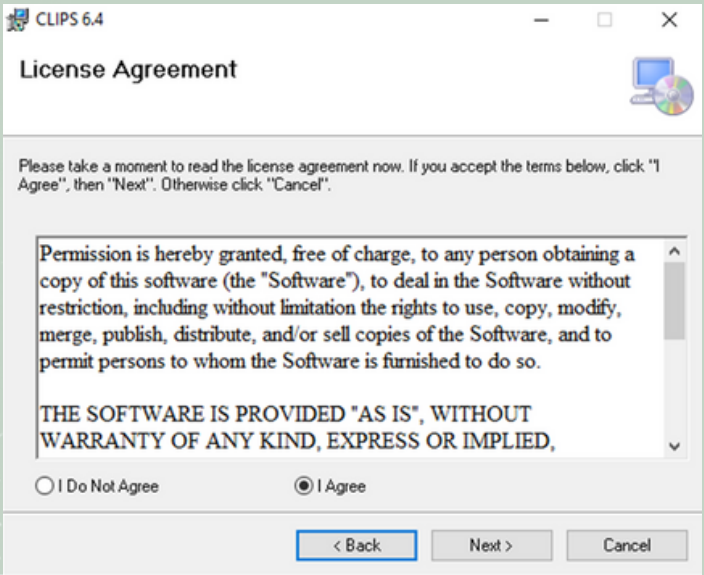
PASSO 3: O download começará automaticamente. Após concluído, basta dar um duplo clique no arquivo executável.



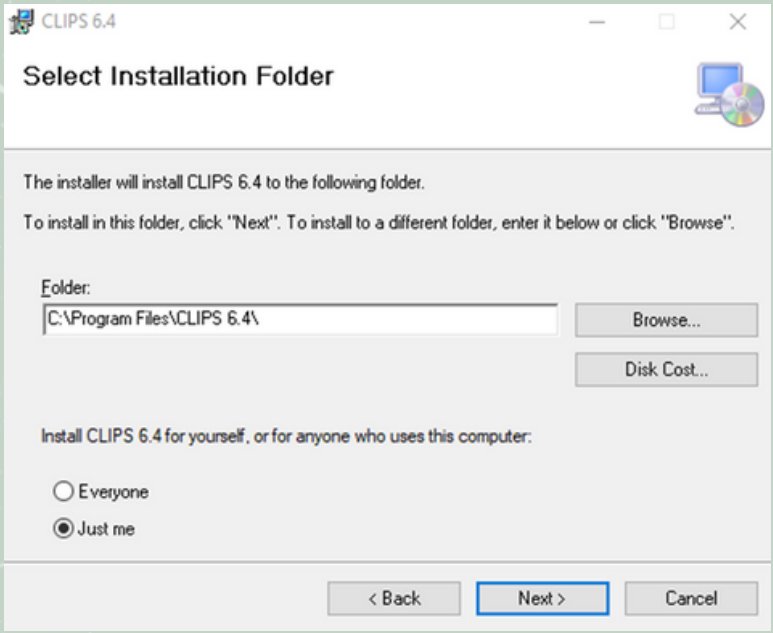
PASSO 4: Após inicializar, a seguinte tela introdutória irá aparecer. Clique em Next.



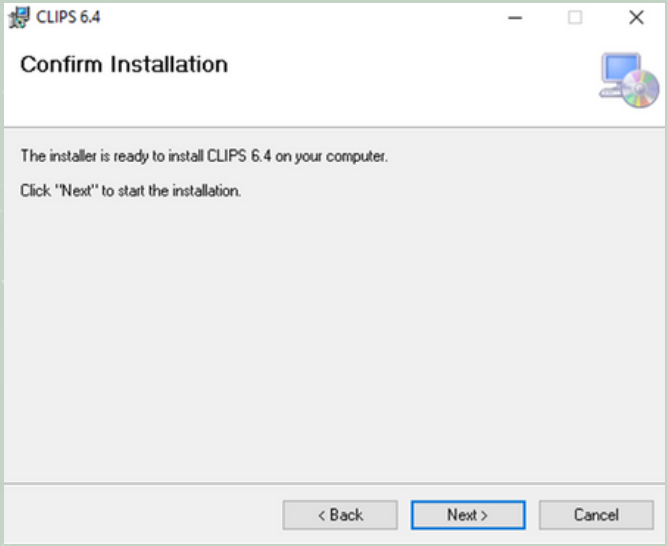
PASSO 5: Para utilizar a IDE, é necessário ler e aceitar o contrato de licença estabelecido pelos desenvolvedores. Clique em “I Agree” e, em seguida, em “Next”.



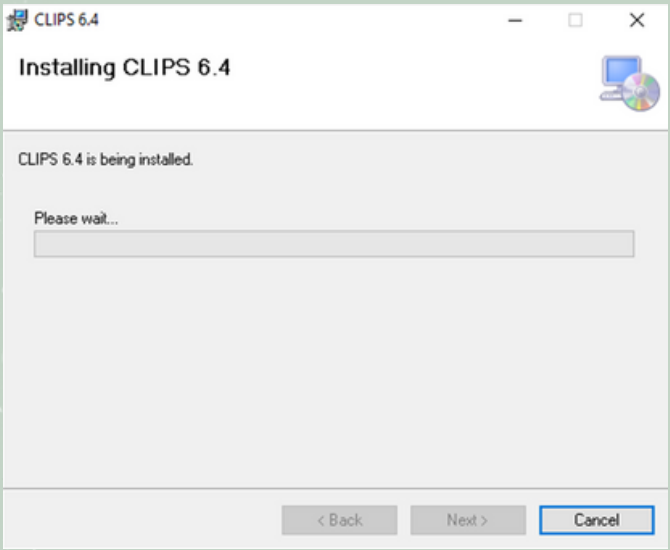
PASSO 6: Aqui, você poderá decidir o local em que o CLIPS será instalado. Previamente, é estabelecido o local C:\Program Files\CLIPS 6.4\, mas é possível fazer alterações nesse caminho clicando em Browse e escolhendo outra pasta. Além disso, você poderá escolher entre instalar o CLIPS apenas para o seu usuário (opção “just me”) ou para todos os usuários do computador (opção “everyone”). Após definido, clique em “Next” para avançar.

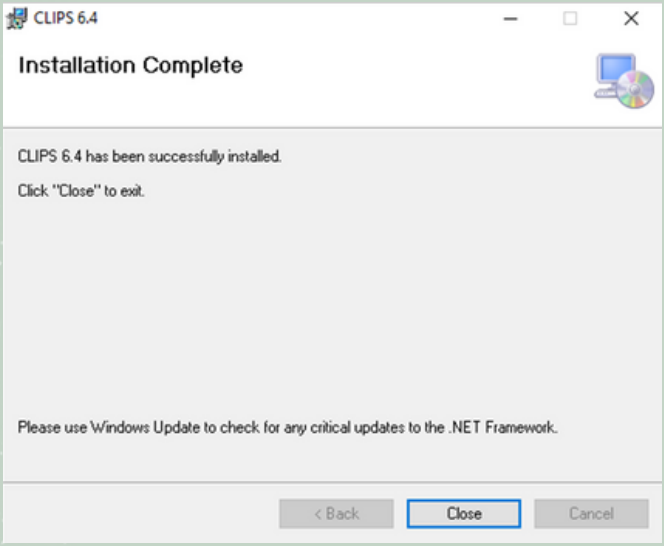


PASSO 7: Agora você está pronto para iniciar a instalação. Clique em “Next” novamente.



PASSO 8: A instalação será iniciada, aguarde a sua conclusão. Após o término, clique em “Close”.

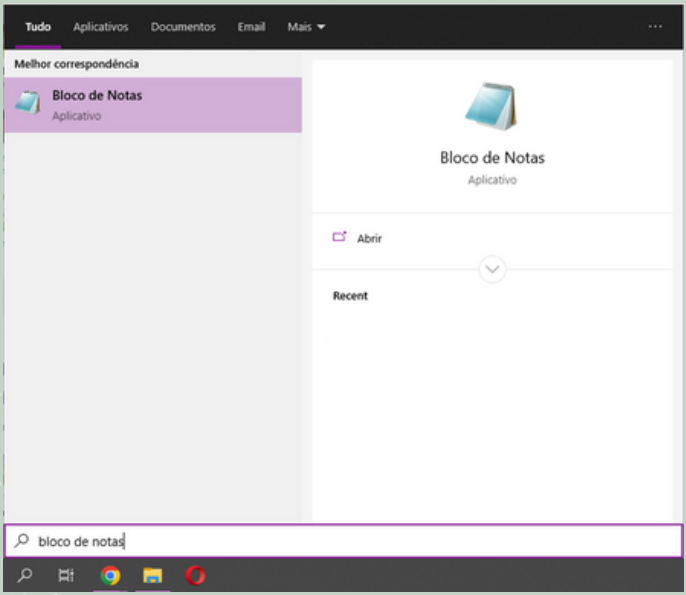




1.4 COMO INICIALIZAR UM PROJETO

"Mas como eu vou programar no CLIPS?"
Você usará a IDE do CLIPS apenas para executar o seu código, mas para desenvolvê-lo será preciso um editor de texto da sua preferência. Como exemplo, utilizaremos o bloco de notas.

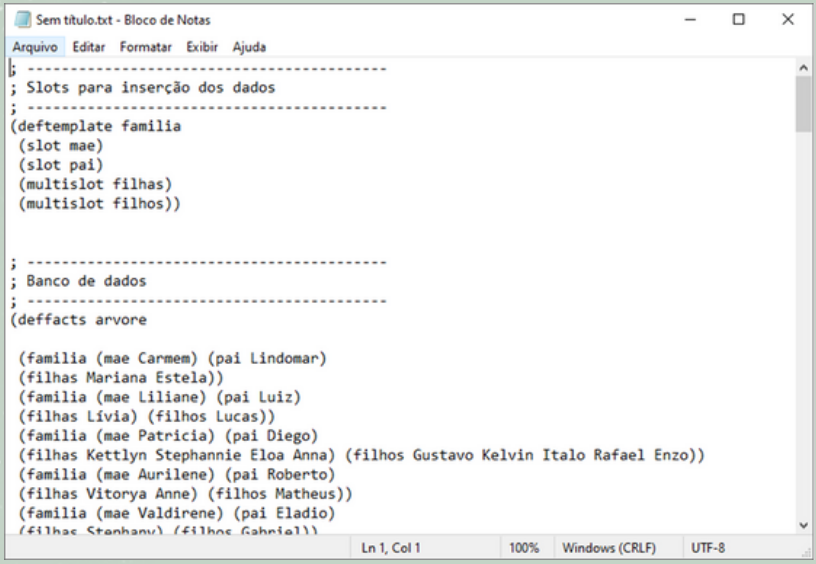
PASSO 1: Pesquise na barra de pesquisa do seu navegador por “bloco de notas”.



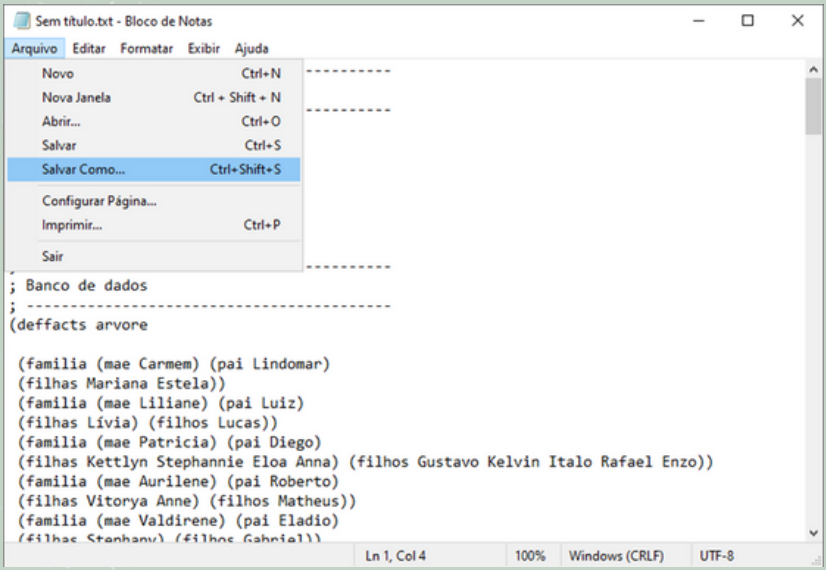
“E como meu código desenvolvido em um bloco de notas funcionará no CLIPS?”

Ao salvar o arquivo, você precisará atribuir a ele a extensão .clp.

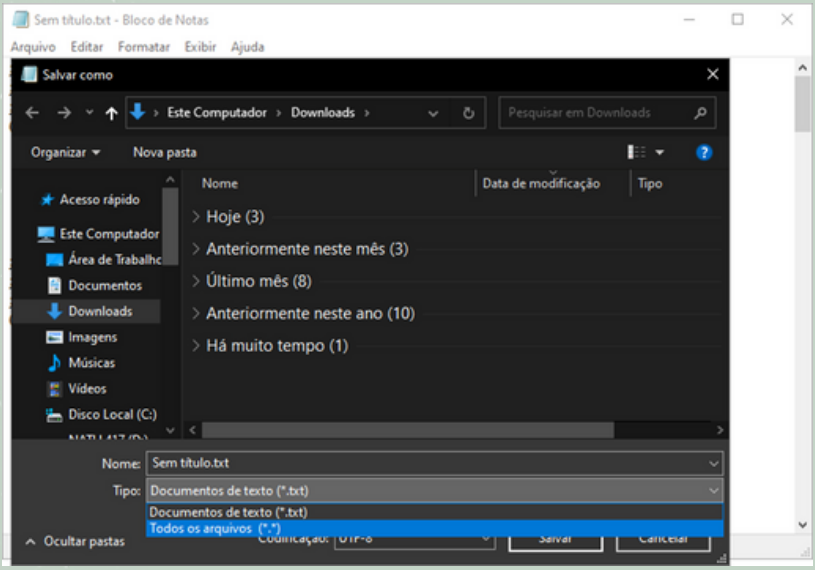
PASSO 2: Clique em “Arquivo”, localizado no menu superior.



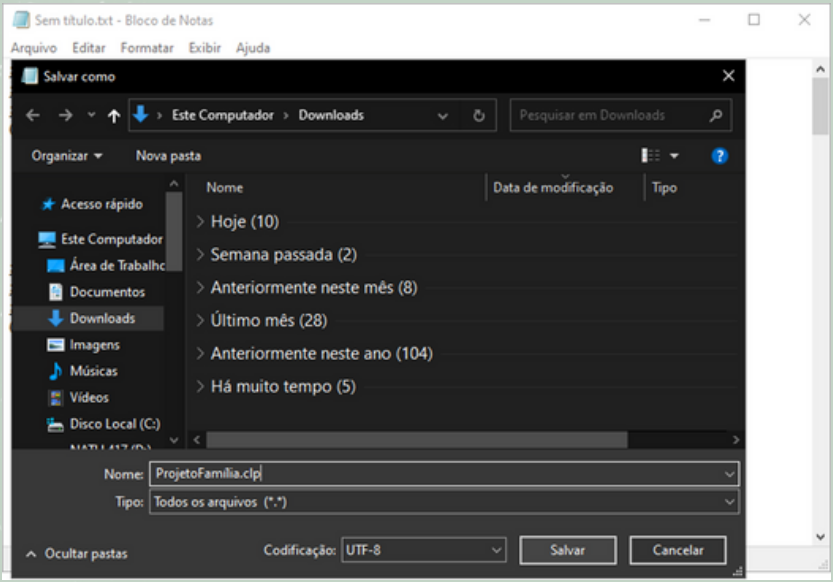
PASSO 3: Vá na opção “Salvar Como...”. Caso prefira, você pode usar o atalho Ctrl + Shift + S.



PASSO 4: Ao escolher um local para salvar, se atente ao tipo de arquivo: selecione “Todos os arquivos (*.*).

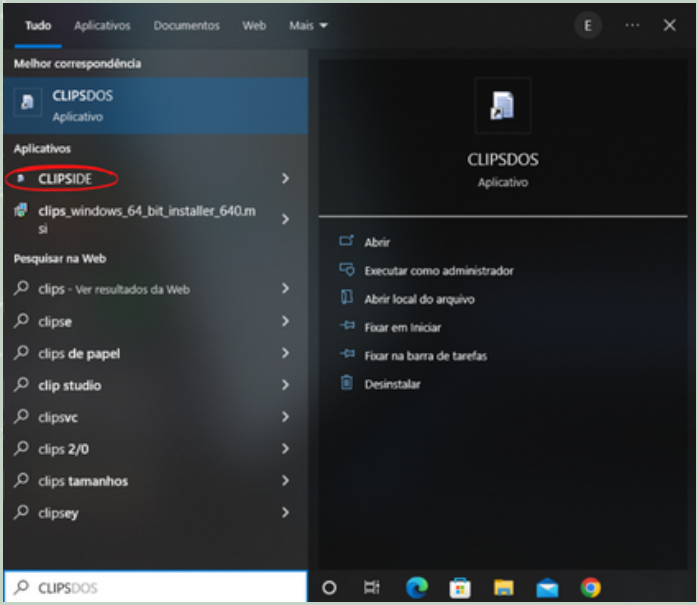


PASSO 5: Nomeie o arquivo, adicionando a extensão .clp ao final.

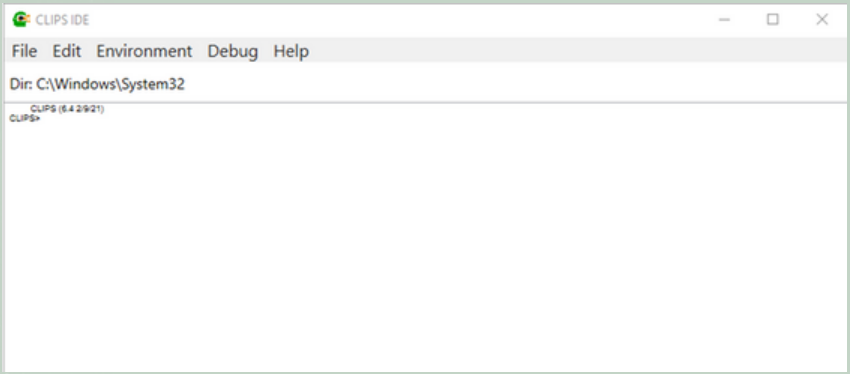


“Agora, como executo?”

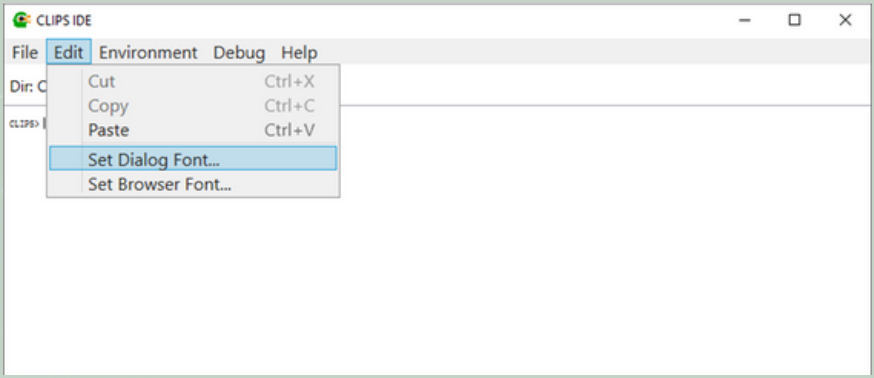
PASSO 1: Na barra de pesquisa do seu sistema operacional, pesquise por CLIPS. Clique em CLIPSIDE.



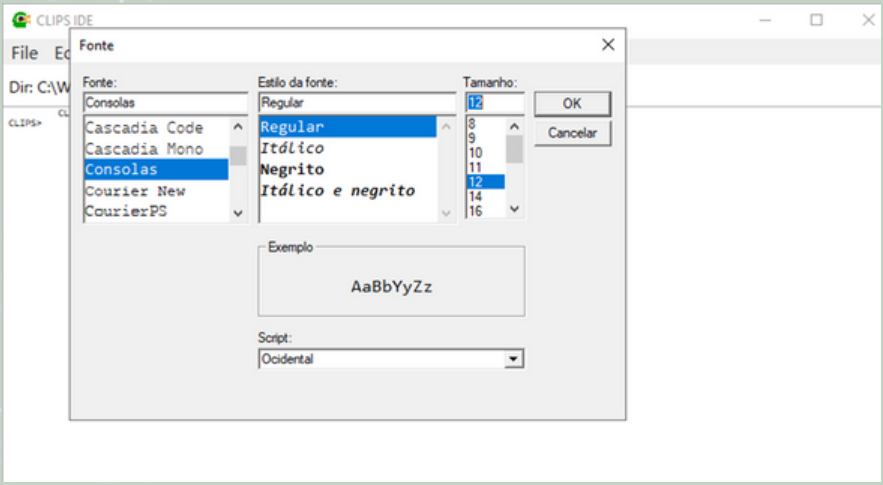
PASSO 2: Esse é o ambiente do CLIPS. Vamos prepará-lo antes.



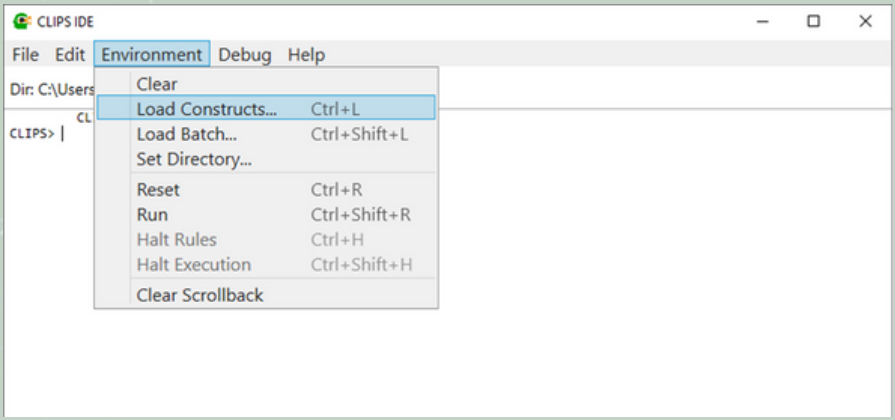
Vamos aumentar a fonte: isso irá te auxiliar a ter uma melhor visualização. Clique em “Edit”, localizado no menu superior. Selecione “Set Dialog Font...”.



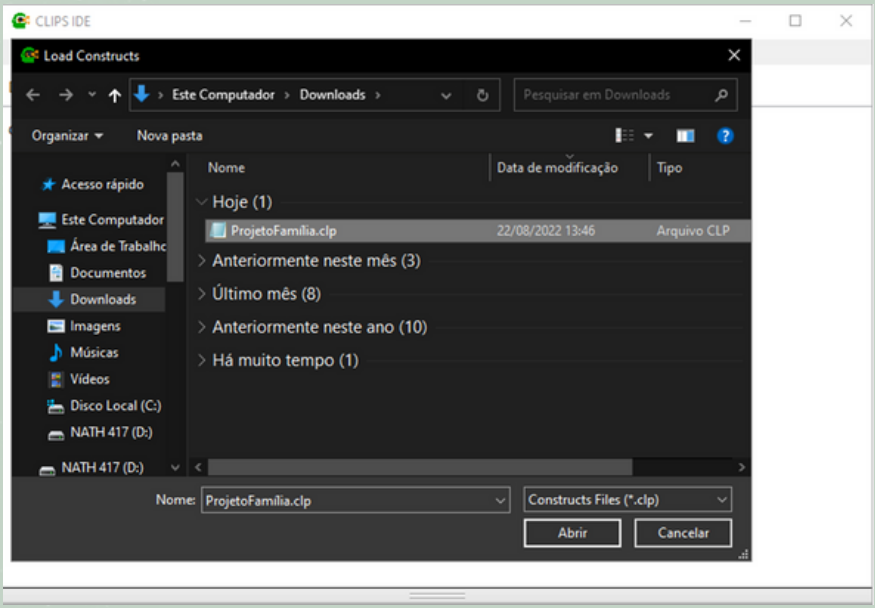
Defina a fonte e o tamanho dela, de acordo com a sua preferência, e aperte em “OK.



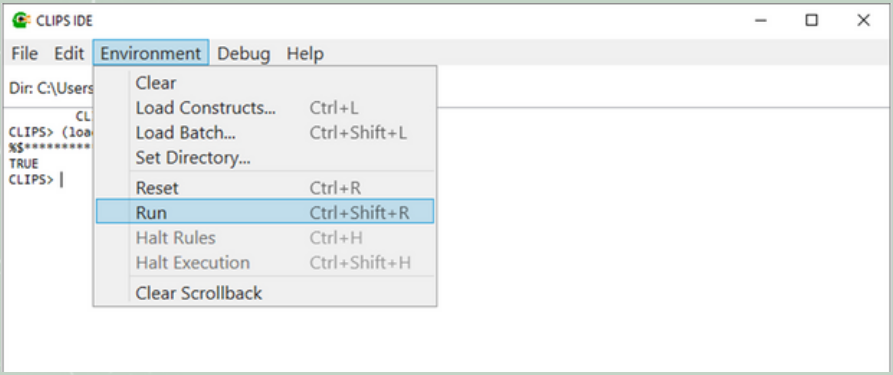
PASSO 3: Para carregar um projeto, na aba “Environment” no menu superior, selecione “Load Constructs...”.



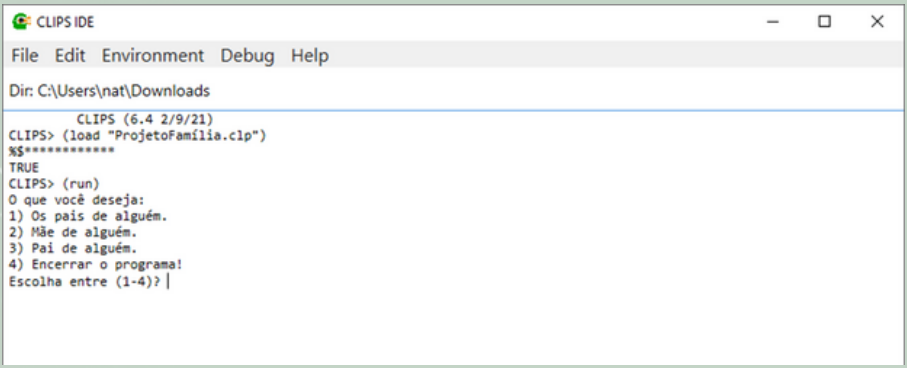
PASSO 4: Selecione o projeto de sua preferência.



PASSO 5: Para executar, vá na mesma aba e clique em “Run”.



Pronto! Seu programa está rodando.



2.CAPÍTULO 2

1.1 CASE 1: FAMÍLIA

Imagem 1: O programa é iniciado e é solicitado que o usuário digite o número da informação que ele quer receber.

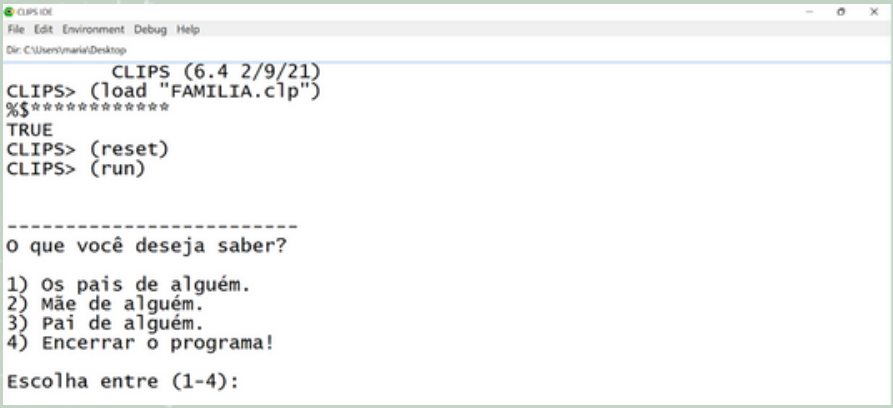


Imagem 2: Quando é digitado a opção 1, o programa pergunta “De quem” o usuário quer saber as informações.

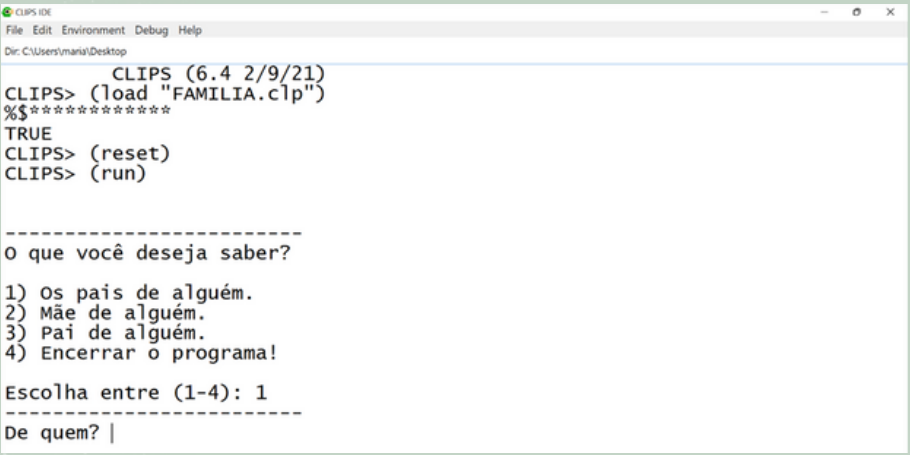


Imagem 3: Então, quando é digitado um nome, que está presente no banco de dados, ele retorna as informações solicitadas. Na opção 1, o programa retornará o nome do pai e da mãe do filho que foi digitado.
Ao final, ele mostra novamente as opções, que chamaremos de MENU a partir de agora, para que o usuário possa continuar a consulta no programa até que a sessão seja finalizada.

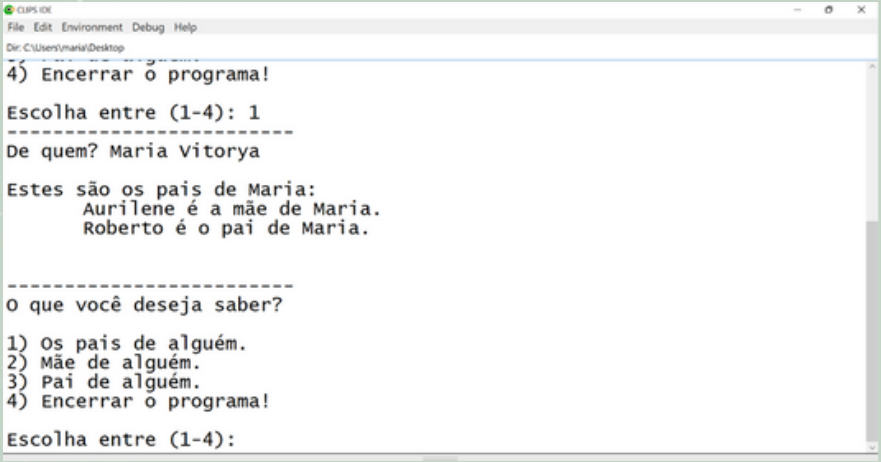


Imagem 4: Agora, o usuário digitou a opção número 2 e novamente o programa pergunta “De quem” o usuário quer saber.

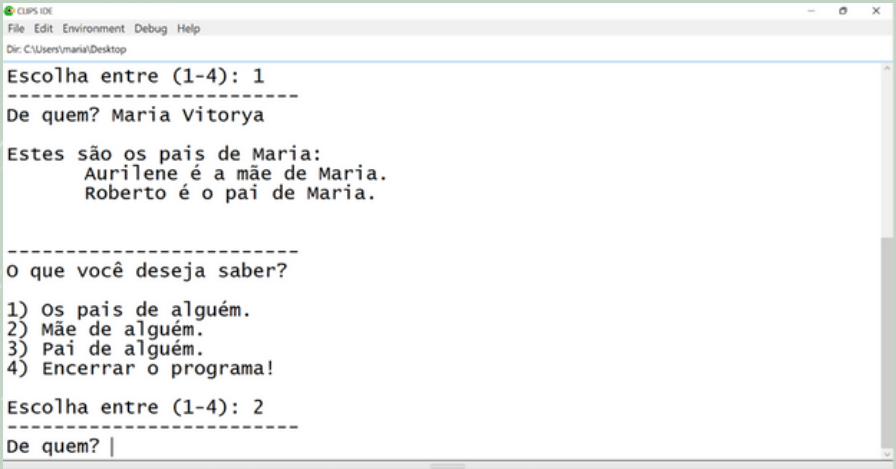


Imagem 5: No caso, o programa retornou o nome da mãe do Lucas, pois está registrado no banco de dados.

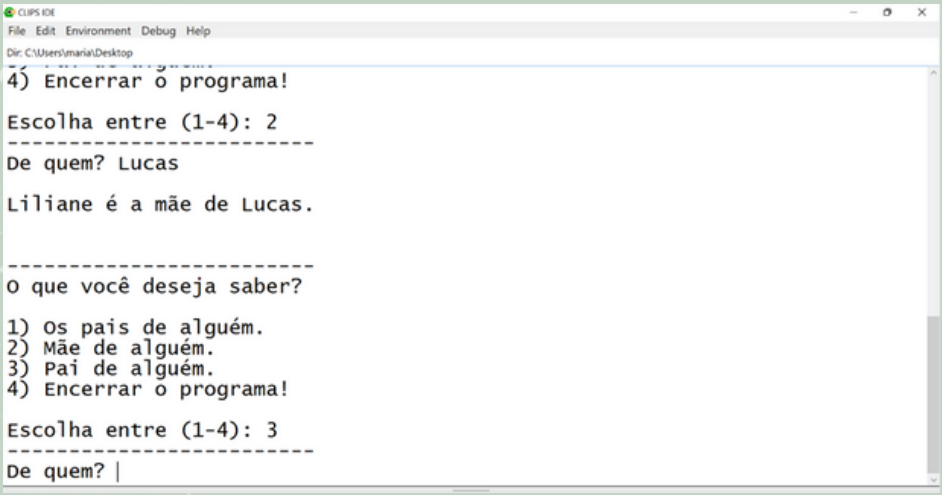


Imagem 6: Então, o menu é mostrado novamente e o usuário digitou a opção 3, que retornará o nome do pai de alguém registrado em nosso banco de dados.

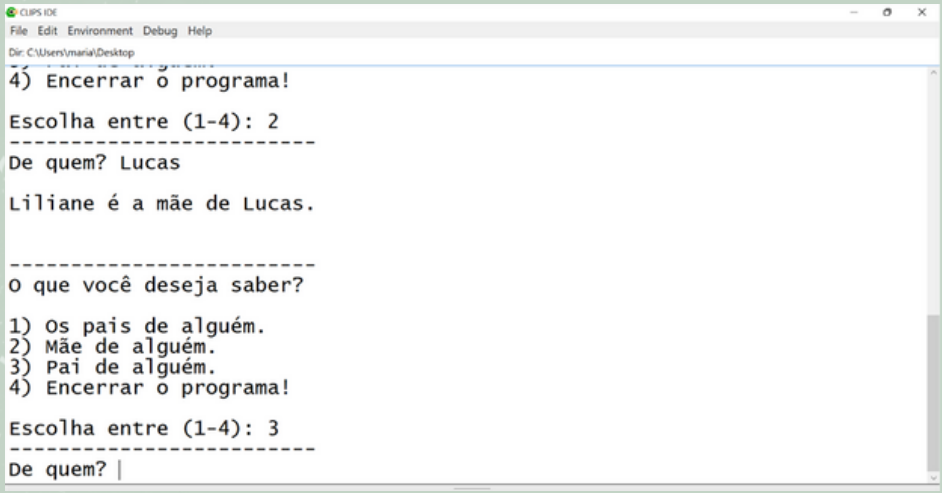


Imagem 7: E aqui está o resultado, o pai da Kettlyn se chama Diego, então, o menu é mostrado novamente.

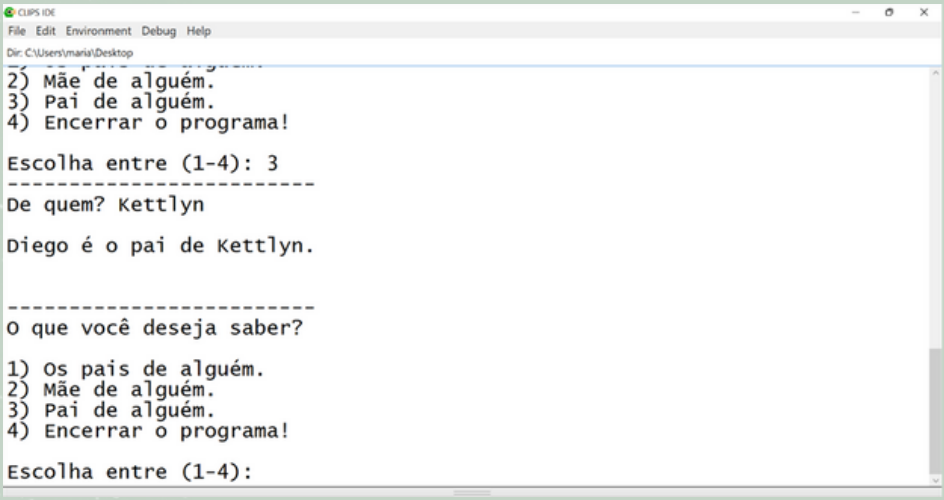


Imagem 8: Nesse caso, quando o usuário digitou um nome que não está registrado no banco de dados, é mostrado que o nome Julia, no caso, não é conhecido.

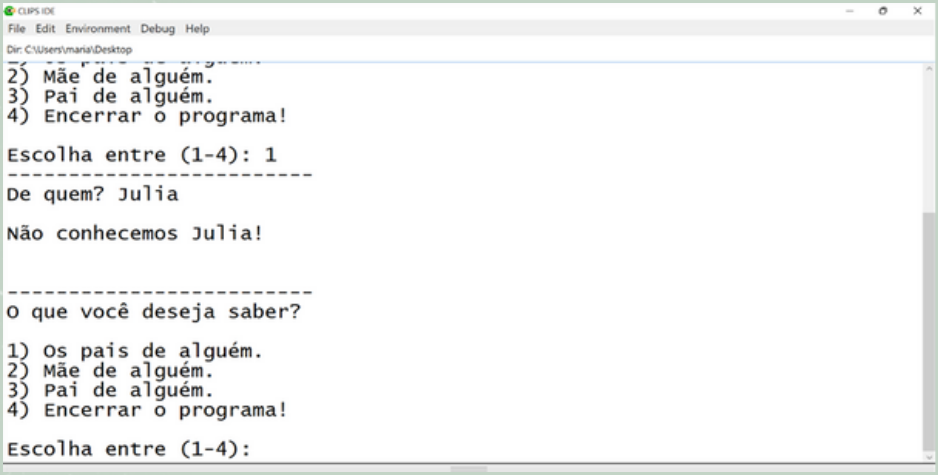


Imagem 9: Agora, quando é digitado o nome de algum pai ou mãe registrado no banco de dados, é mostrado que não há informações sobre os pais dele.

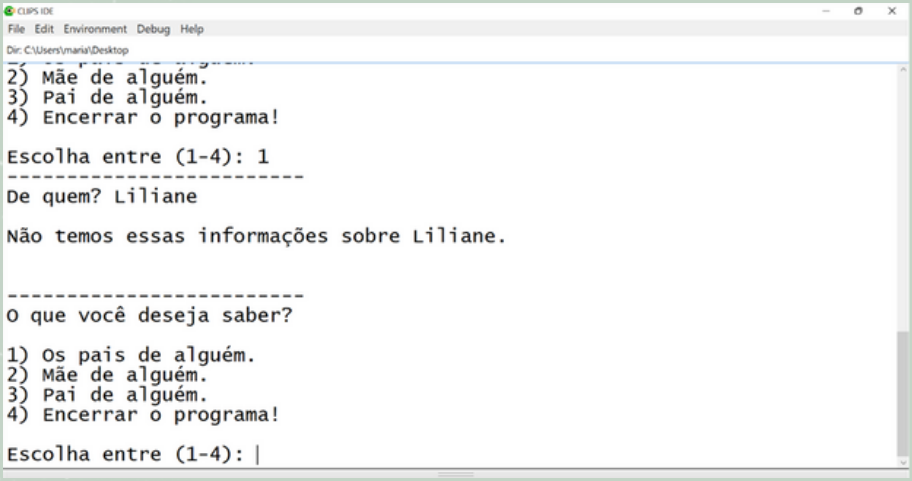


Imagem 10: Agora, quando o usuário digita uma opção que não está entre 1 e 4, é mostrado que a opção é inválida e solicita que o usuário digite outra opção.

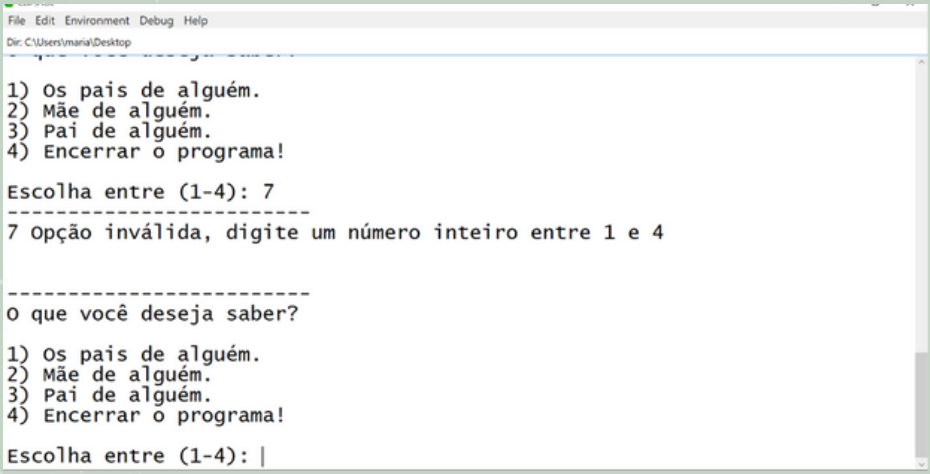
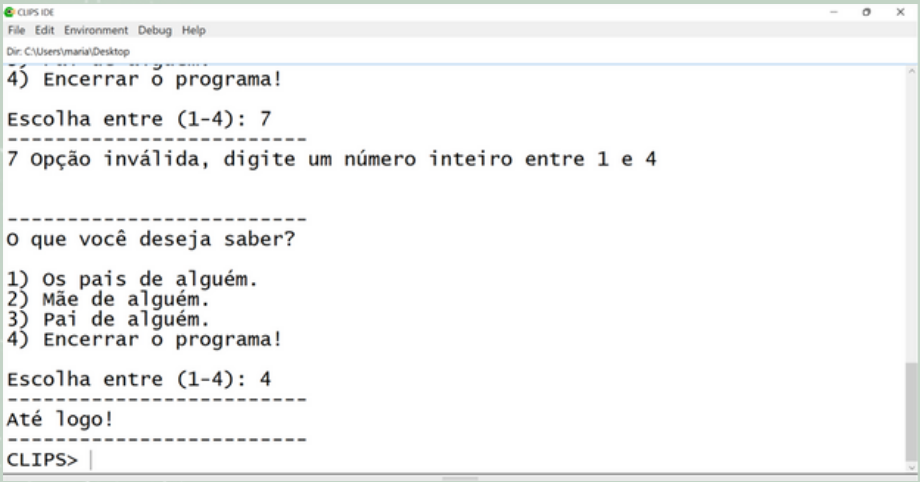


Imagem 11: Agora quando o usuário digita a opção 4, o programa é encerrado.



“Hello World” com inteligência artificial - Árvore genealógica

Primeiramente, iremos criar os campos que iremos inserir os dados:

(deftemplate familia (slot mae) (slot pai) (multislot filhas) (multislot filhos))

Agora iremos criar o banco de dados:

```
(deffacts arvore
 (familia (mae Carmem) (pai Lindomar)
 (filhas Mariana Estela))
 (familia (mae Liliane) (pai Luiz)
 (filhas Lívia) (filhos Lucas))
 (familia (mae Patricia) (pai Diego)
 (filhas Kettlyn Stephannie Eloa Anna) (filhos Gustavo Kelvin Italo Rafael Enzo))
 (familia (mae Aurilene) (pai Roberto)
 (filhas Maria Vitorya Anne) (filhos Matheus))
 (familia (mae Valdirene) (pai Eladio)
 (filhas Stephany) (filhos Gabriel))
 (familia (mae Cyllene) (pai Robson)
 (filhas Leticia Isabelle)))
```

Agora iremos criar a regra para chamar a inicialização do programa:

```
(defrule inicio => (assert (fase Fase-de-escolha)))
```

Agora criaremos um pequeno menu para o programa:

```
(defrule menu
        (fase Fase-de-escolha)
=>
(printout t "O que você deseja:
1) Os pais de alguém.
2) Mãe de alguém.
3) Pai de alguém.
4) Encerrar o programa!
Escolha entre (1-4)? ")
        (assert (usuario-select (read))))
```

Escrevendo uma opção válida, solicitaremos o nome:

```
(defrule opcao-correta
        ?fase <- (fase Fase-de-escolha)
        ?opcao <- (usuario-select ?select&1 | 2 | 3 | 4)
=>
        (retract ?fase ?opcao)
        (assert (selecao ?select))
        (assert (fase select-crianca)))
```

Caso o usuário digite um número inválido, solicita uma nova digitação:

```
(defrule opcao-incorreta
        ?fase <- (fase Fase-de-escolha)
        ?opcao <- (usuario-select ?select&~1&~2&~3&~4)
=>
        (retract ?fase ?opcao)
        (assert (fase Fase-de-escolha))
            (printout t ?select " Opção inválida, digite um número
inteiro entre 1 e 4" crlf))
```

Digita o nome e chama a procura pela criança:

```
(defrule get-nome
 ?fase <- (fase select-crianca)
 ?opcao <- (selecao ?select&1 | 2 | 3)
=>
        (retract ?fase)
        (assert (fase select-parente))
        (printout t "De quem? ")
        (assert (get-alguem (read))))
```

Digitação de um nome válido chama o “return” da opção selecionada:

```
(defrule correct-nome
        ?fase <- (fase select-parente)
        ?nome <- (get-alguem ?pessoa)
        (or (familia (filhas $? ?pessoa $?))
        (familia (filhos $? ?pessoa $?)))
=>
        (retract ?fase)
        (assert (fase return-pais)))
```

Mostra as informações da opção 1:

```
(defrule get-pais
        ?fase <- (fase return-pais)
        ?nome <- (get-alguem ?pessoa)
        ?opcao <- (selecao ?select&1)
          (and (or (familia (mae ?mamae) (pai ?papai) (filhos $? ?
pessoa $?))
           (familia (mae ?mamae) (pai ?papai) (filhas $? ?pessoa
$?))))
=>
        (retract ?fase ?nome ?opcao)
        (assert (fase Fase-de-escolha))
        (printout t "Estes são os pais de " ?pessoa ":
        " ?mamae " é a mãe de " ?pessoa ".
        " ?papai " é o pai de " ?pessoa "." crlf))
```

Mostra as informações da opção 2:

```
(defrule get-mae
 ?fase <- (fase return-pais)
 ?nome <- (get-alguem ?pessoa)

 ?opcao <- (selecao ?select&2)
          (and (or (familia (mae ?mamae) (pai ?papai) (filhos $? ?
pessoa $?))
           (familia (mae ?mamae) (pai ?papai) (filhas $? ?pessoa
$?))))
=>
        (retract ?fase ?nome ?opcao)
        (assert (fase Fase-de-escolha))
        (printout t ?mamae " é a mãe de " ?pessoa "." crlf))
```

Mostra as informações da opção 3:

```
(defrule get-pai
        ?fase <- (fase return-pais)
        ?nome <- (get-alguem ?pessoa)
        ?opcao <- (selecao ?select&3)
          (and (or (familia (mae ?mamae) (pai ?papai) (filhos $? ?
pessoa $?))
           (familia (mae ?mamae) (pai ?papai) (filhas $? ?pessoa
$?))))
=>
        (retract ?fase ?nome ?opcao)
        (assert (fase Fase-de-escolha))
        (printout t ?papai " é o pai de " ?pessoa "." crlf))
```

Caso o programa não saiba quem são os pais de alguém, retorna para o menu:

```
(defrule sem-info
        ?fase <- (fase select-parente)
        ?nome <- (get-alguem ?pessoa)
        ?opcao <- (selecao ?select)
        (and (or (familia (mae ?pessoa))
        (familia (pai ?pessoa))))
```

```
=>
 (retract ?fase ?nome ?opcao)
 (assert (fase Fase-de-escolha))
  (printout t "Não temos essas informações sobre " ? pessoa
 "." crlf))

(defrule nome-errado
        ?fase <- (fase select-parente)
        ?nome <- (get-alguem ?pessoa)
        ?opcao <- (selecao ?select)
              (familia (mae ?mamae) (pai ?papai) (filhas $? menina) (filhos $?menino))
=>
        (retract ?fase ?nome ?opcao)
        (assert (fase Fase-de-escolha))
        (printout t "Não conhecemos " ?pessoa "!" crlf))
```

Opção 4 encerra o programa:

```
(defrule sair
        ?fase <- (fase select-crianca)
        ?opcao <- (selecao ?select&4)
=>
        (retract ?fase ?opcao)
        (printout t "Até logo!" crlf))
```

Para executar, basta carregar o programa, basta selecionar no clips a opção Enviroment → Load Constructs e selecionar o programa
Após carregar o programa, digitar “(reset)” e “(run)”

2.2 CASE 2: TESTANDO TEMPERATURA

Imagem 1:Quando o programa é iniciado, é mostrado o menu de opções para consultas que o usuário pode fazer

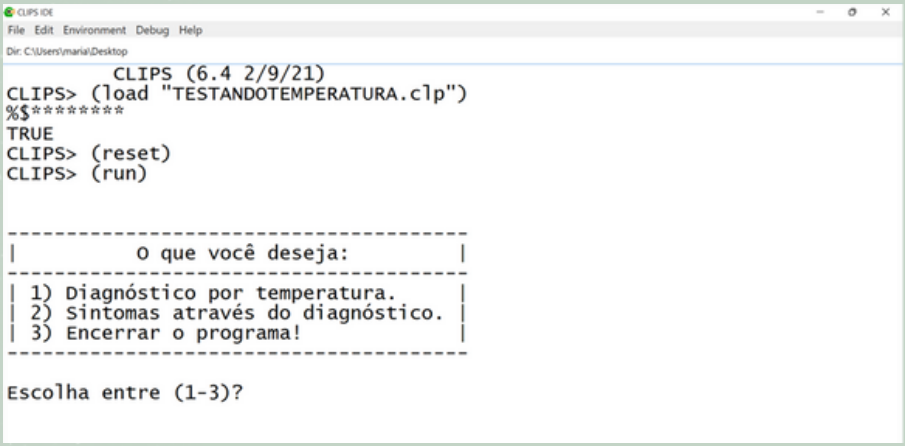


Imagem 2: Quando o usuário digita a opção 1, é solicitado a temperatura do usuário para que seja retornado o diagnóstico.

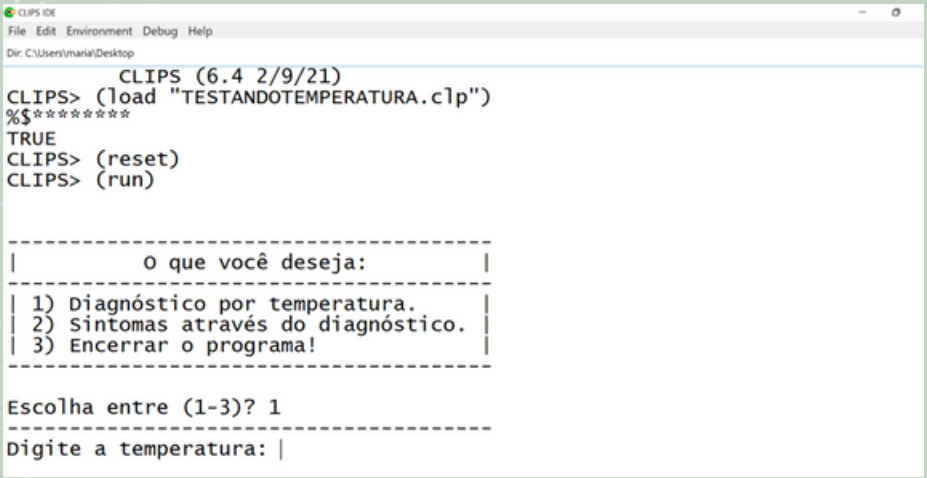


Imagem 3: E aqui está o resultado, quando é digitado a temperatura 34, é previsível que o diagnóstico seja Hipotermia.

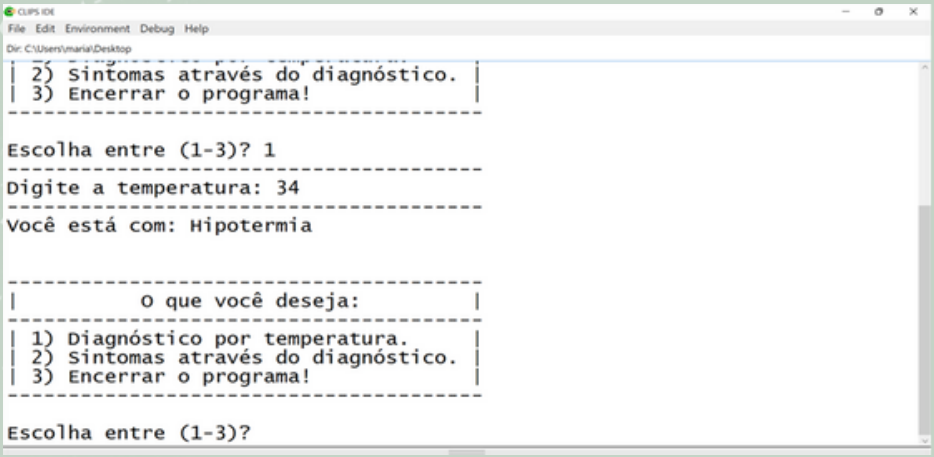


Imagem 4: Aqui é outro resultado, quando é digitado a temperatura 36, é previsível que o diagnóstico seja de Temperatura Normal.

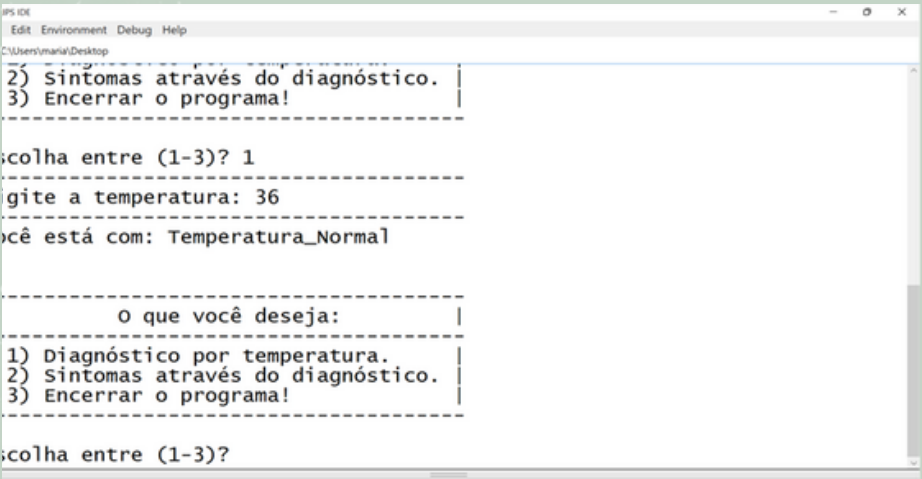


Imagem 5: Agora, quando o usuário quer saber os sintomas através de um diagnóstico, é digitado a opção 2 e é mostrado os diagnósticos que ele pode consultar.

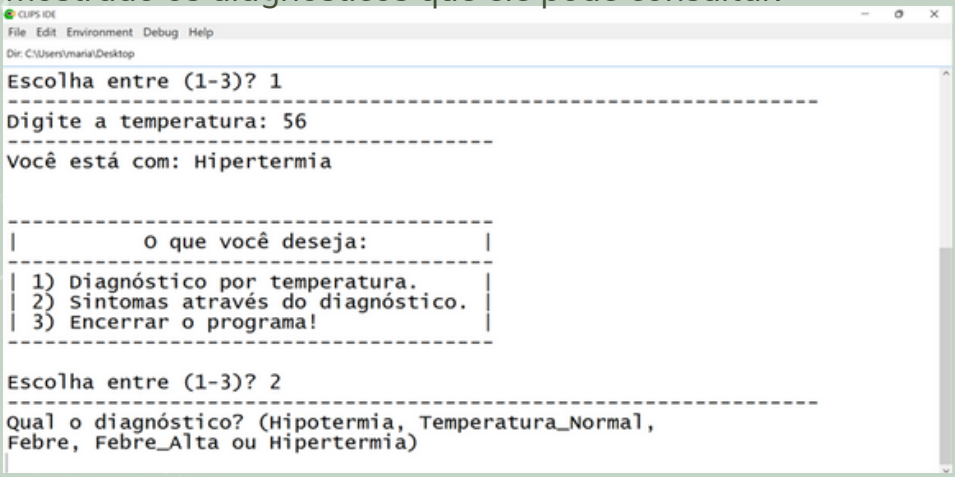


Imagem 6: E pronto, os sintomas relacionados ao diagnóstico estão aqui. E como toda vez, o menu é mostrado para o usuário continuar a consulta, a não ser que ele termine a sessão.

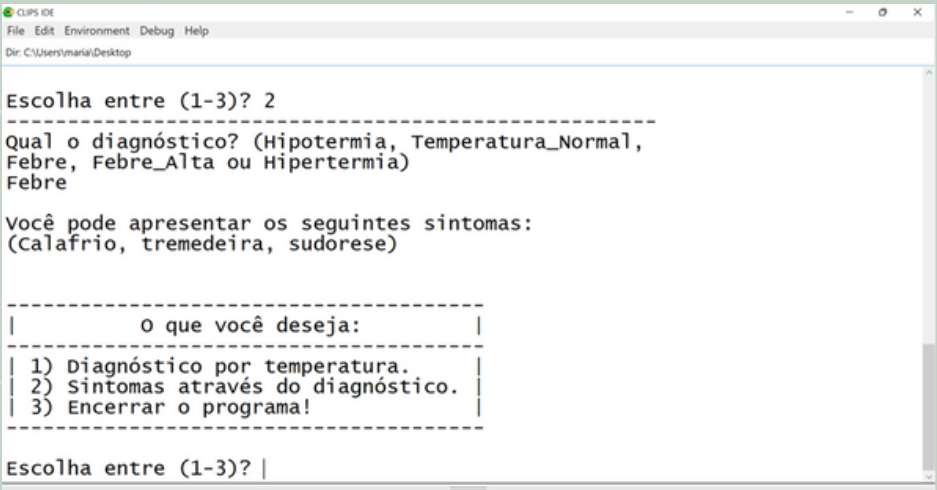


Imagem 7: Agora, quando o usuário digite um número fora do limite estipulado, assim como no case anterior, é mostrado que a opção é inválida e o menu aparece novamente.

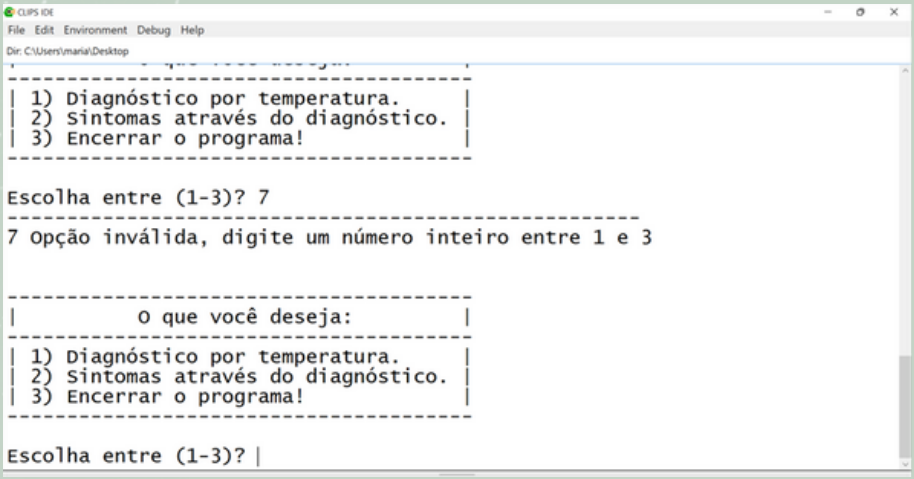
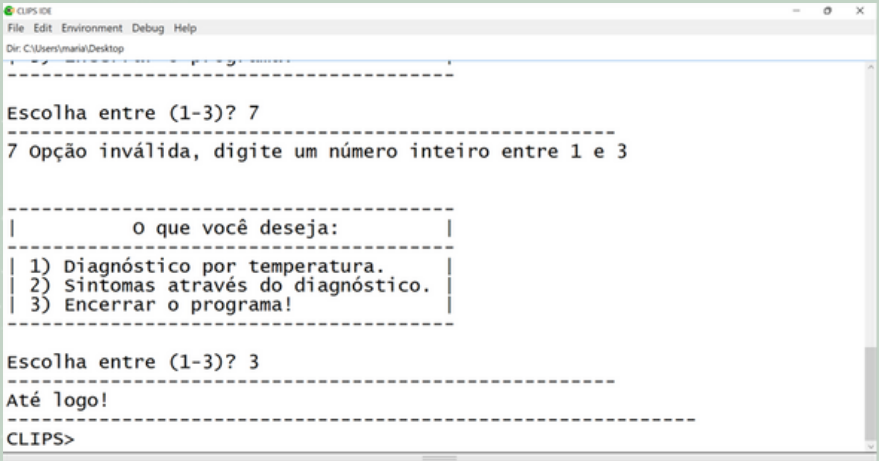


Imagem 8: E a opção 3 termina a sessão, assim como esse tutorial. ATÉ LOGO!



Teste de temperatura com diagnóstico e sintomas

Primeiramente, vamos criar os campos que guardarão os dados:

(deftemplate temperatura (slot diagnostico) (multislot sintomas))

Agora vamos criar o banco de dados:

```
(deffacts dados
 (temperatura (diagnostico Hipotermia)
  (sintomas Tremores, hipóxia, dormência, cansaço, lentargia))
 (temperatura (diagnostico Temperatura_Normal)
 (sintomas Nenhum))
 (temperatura (diagnostico Febre)
 (sintomas Calafrio, tremedeira, sudorese))
 (temperatura (diagnostico Febre_Alta)
    (sintomas Dores_musculares, cefaleia, apatia, indisposição, sudorese,
inapertência, desidratação, irritabilidade, confusão))
 (temperatura (diagnostico Hipertermia)
  (sintomas Sudorese, cefaleia, lipotimia, fraqueza, cãibras, alucinações,
convulsões, hipotensão, bradipneia, desmaios, náuseas, vômitos)))
```

Depois do banco de dados criado, vamos criar a regra para chamar a inicialização do programa:

```
(defrule inicio => (assert (fase Fase-de-escolha)))
```

Agora montaremos um pequeno menu para o programa:

```
(defrule menu
        (fase Fase-de-escolha)
```

```
=>
(printout t "
--------------------------------------
|       O que você deseja:       |
--------------------------------------
| 1) Diagnóstico por temperatura.    |
| 2) Sintomas através do diagnóstico. |
| 3) Encerrar o programa!          |
--------------------------------------
Escolha entre (1-3)? ")
        (assert (usuario-select (read)))
(printout t "-----------------------------------------------
----" crlf)
)
```

Caso o usuário digite um número inválido, que esteja fora do intervalo de 1 e 3, solicita uma nova digitação:

```
(defrule opcao-incorreta
        ?fase <- (fase Fase-de-escolha)
        ?opcao <- (usuario-select ?select&~1&~2&~3)
=>
        (retract ?fase ?opcao)
        (assert (fase Fase-de-escolha))
            (printout t ?select " Opção inválida, digite um
número inteiro entre 1 e 3" crlf))
```

Escrevendo uma opção válida, solicitaremos o nome:

```
(defrule opcao-correta
        ?fase <- (fase Fase-de-escolha)
        ?opcao <- (usuario-select ?select&1 | 2 | 3)
=>
        (retract ?fase ?opcao)
        (assert (selecao ?select))
        (assert (fase select-problema)))
```

Se a opção escolhida foi a número 1, ele pede para que o usuário digite a temperatura:

```
(defrule get-temperatura
        ?fase <- (fase select-problema)
        ?opcao <- (selecao ?select&1)
=>
        (retract ?fase ?opcao)
        (assert (fase select-temp))
        (printout t "Digite a temperatura: ")
        (bind ?temp (read))
(printout t "-------------------------------------" crlf)
```

A partir daqui começam as condicionais de temperatura:

```
        (if (<= ?temp 35)
        then
        (bind ?diagnostico Hipotermia)

        (printout t "Você está com: " ?diagnostico crlf)

        else
        (if (<= ?temp 37.5)
        then
        (bind ?diagnostico Temperatura_Normal)

        (printout t "Você está com: " ?diagnostico crlf)

        else
        (if (<= ?temp 39.5)
        then
        (bind ?diagnostico Febre)

        (printout t "Você está com: " ?diagnostico crlf)
```

```
else
 (if (<= ?temp 41)
 then
 (bind ?diagnostico Febre_Alta)

(printout t "Você está com: " ?diagnostico crlf)

 else

 (bind ?diagnostico Hipertermia)
 (printout t "Você está com: " ?diagnostico crlf))))
)
 (assert (fase Fase-de-escolha))
)
```

Caso a opção escolhida for o número 2, ele pede para o usuário informar o diagnóstico. De preferência, o que foi informado na opção 1:

```
(defrule get-diagnostico
        ?fase <- (fase select-problema)
        ?opcao <- (selecao ?select&2)
=>
        (retract ?fase)
        (assert (fase select-sinto))
                (printout t "Qual o diagnóstico? (Hipotermia,
Temperatura_Normal,
Febre, Febre_Alta ou Hipertermia)" crlf)
        (assert (get-diagnostico (read)))
)
```

Depois de saber o diagnóstico, o sistema busca no banco de dados os sintomas relacionados à esse diagnóstico e exibe para o usuário:

```
(defrule get-doenca
 ?fase <- (fase select-sinto)
 ?opcao <- (selecao ?select&2)
```

```
?diagnostico <- (get-diagnostico ?doenca)
   (temperatura (diagnostico ?doenca) (sintomas $? sintoma))
=>
 (retract ?fase ?opcao ?diagnostico)
 (printout t "

Você pode apresentar os seguintes sintomas: " crlf ? sintoma crlf)

        (assert (fase Fase-de-escolha))
)

 E a opção 3 encerra o programa:

(defrule sair
        ?fase <- (fase select-problema)
        ?opcao <- (selecao ?select&3)
=>
        (retract ?fase ?opcao)
        (printout t "Até logo!
------------------------------------------------------------"
crlf)
)
```

2.3 CASE 3: SELETOR DE VINHOS

Imagem 1: Para iniciar o programa, digite (load + caminho do arquivo), (reset) e (run)

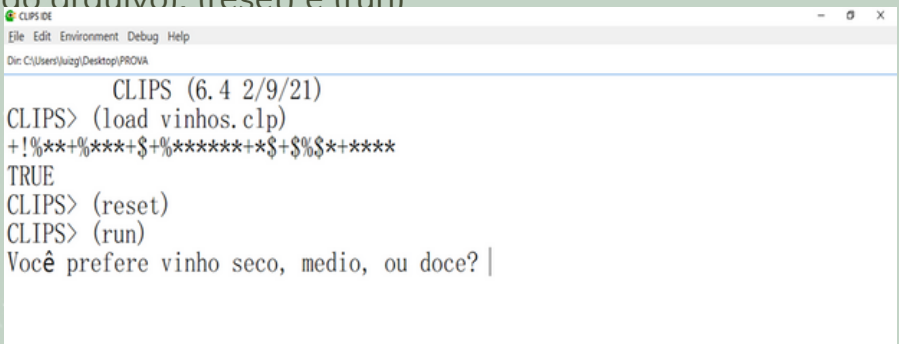


Imagem 2: Selecione o tipo de vinho de sua preferência

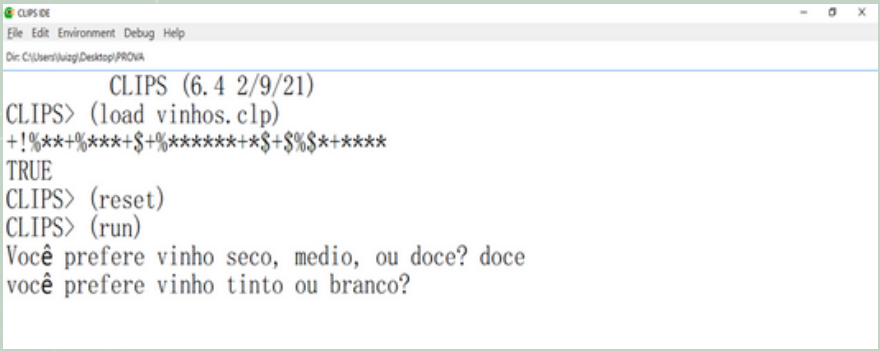


Imagem 3: Escolha entre vinho tinto ou branco

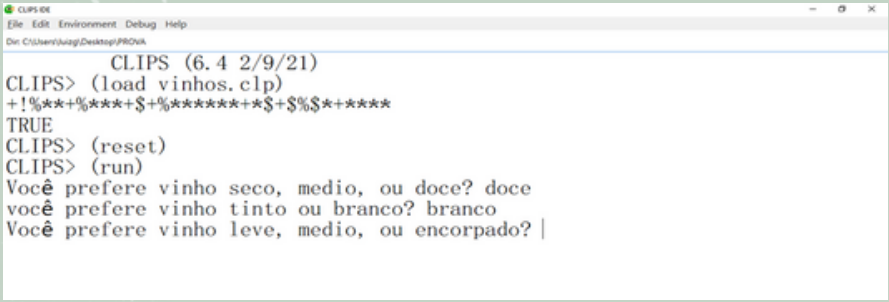


Imagem 4: Defina a potência do vinho

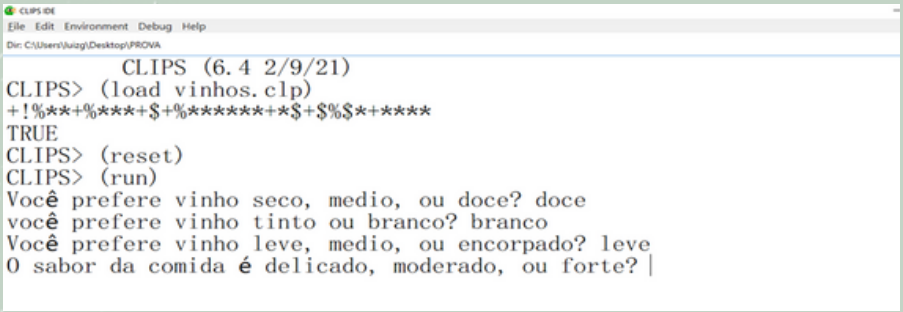


Imagem 5: Informe o sabor da comida

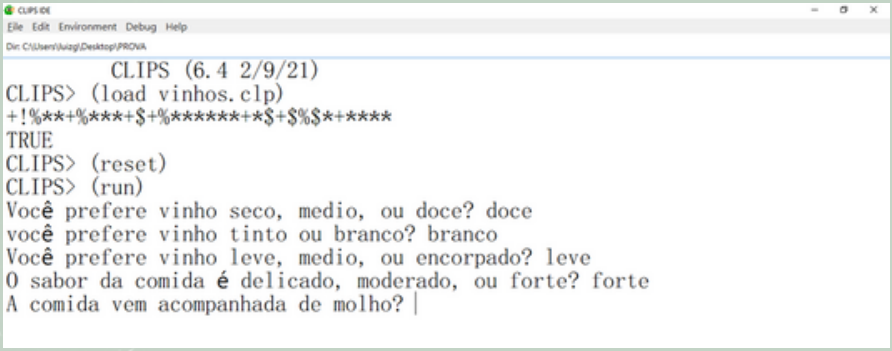


Imagem 6: Diga se tem molho na comida (caso não, não haverá a pergunta do tipo)

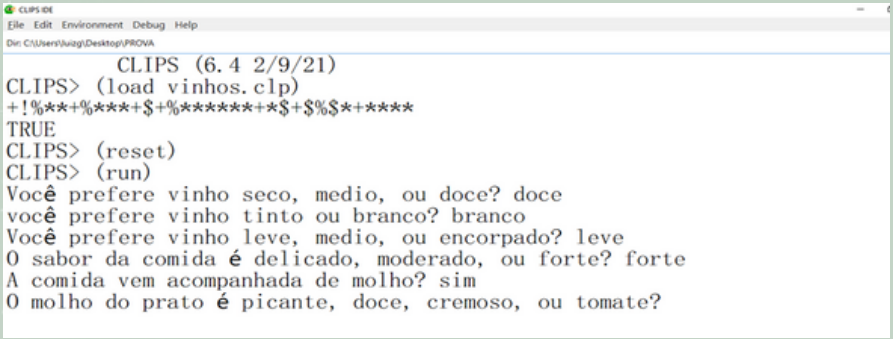


Imagem 7: Informe o tipo de molho

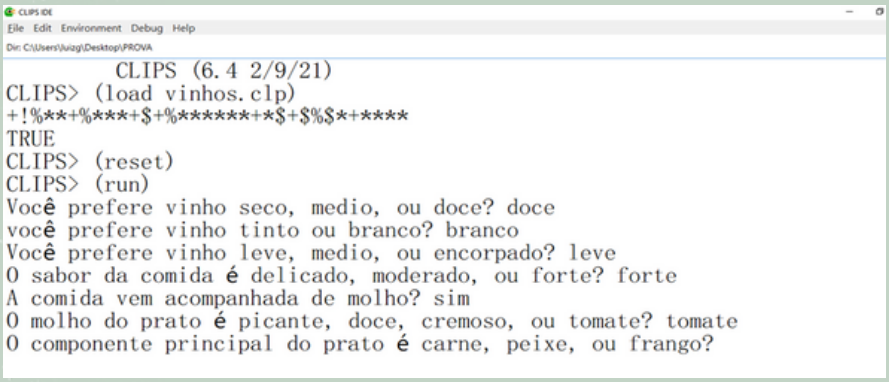


Imagem 8: Diga qual o componente principal do prato

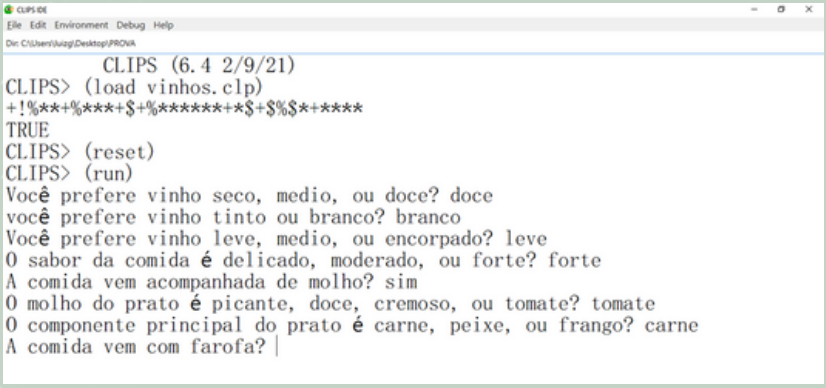
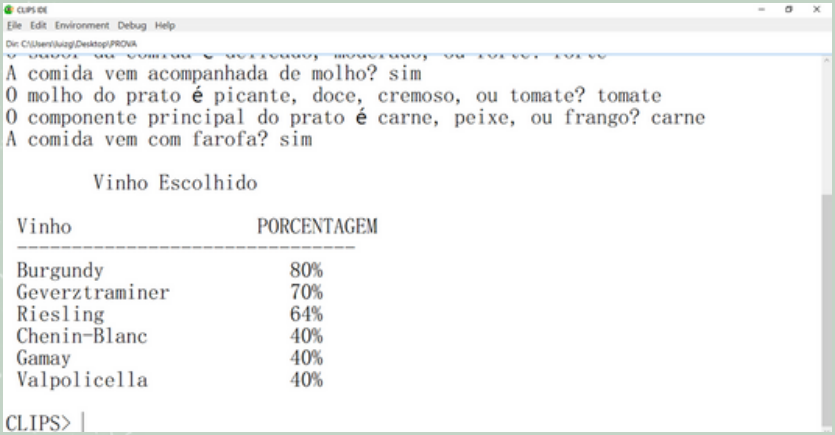


Imagem 9: Informe se há farofa no prato



Observação: Cada escolha diferente altera as porcentagens e as melhores escolhas. São mostradas apenas aquelas com porcentagem maior que 20.

Seletor de Vinhos por prato

Inicialmente criaremos um módulo e definiremos uma função principal:

```
(defmodule MAIN (export ?ALL))
(deffunction MAIN::Faz-Questoes (?pergunta ?allowed-valores)
(print ?pergunta)
(bind ?resposta (read))
(if (lexemep ?resposta) then (bind ?resposta (lowcase ?resposta)))
(while (not (member$ ?resposta ?allowed-valores)) do
(print ?pergunta)
(bind ?resposta (read))
(if (lexemep ?resposta) then (bind ?resposta (lowcase ?resposta))))?resposta)
```

Declarar as variáveis principais do programa e regras iniciais:

```
(deftemplate MAIN::atributos
(slot nome)
(slot valor)
(slot porcentagem (default 100.0)))

(defrule MAIN::start
(declare (salience 10000))
 =>
(set-fact-duplication TRUE)
(focus PERGUNTAS ESCOLHA-QUALIDADES VINHOS MOSTRA-RESULTADO))


(defrule MAIN::transformador-porcentagens ""
```

```
(declare (salience 100)
(auto-focus TRUE))
?rem1 <- (atributos (nome ?rel) (valor ?val)
(porcentagem ?per1))
?rem2 <- (atributos (nome ?rel) (valor ?val)
(porcentagem ?per2))
(test (neq ?rem1 ?rem2))
=>
(retract ?rem1)
(modify ?rem2 (porcentagem (/ (- (* 100 (+ ?per1 ?per2))
(* ?per1 ?per2)) 100))))
```

Definiremos o módulo de perguntas, o qual importará informações do módulo principal:

```
(defmodule PERGUNTAS (import MAIN ?ALL) (export ?
ALL))
```

Declararemos as variáveis principais das perguntas:

```
(deftemplate PERGUNTAS::pergunta
  (slot atributos (default ?NONE))
  (slot questao (default ?NONE))
  (multislot respostas-validas (default ?NONE))
  (slot already-asked (default FALSE))
  (multislot precursors (default ?DERIVE)))
```

Definiremos então as regras e a parte lógica das perguntas:

```
(defrule PERGUNTAS::ask-a-pergunta
  ?f <- (pergunta (already-asked FALSE)
                  (precursors)
                  (questao ?questao)
                  (atributos ?the-atributos)
                  (respostas-validas $?respostas-validas))
```

```
=>
 (modify ?f (already-asked TRUE))
 (assert (atributos (nome ?the-atributos)
 (valor (Faz-Questoes ?questao ?respostas-validas)))))

(defrule PERGUNTAS::precursor-is-satisfied
 ?f <- (pergunta (already-asked FALSE)
 (precursors ?nome is ?valor $?rest))
 (atributos (nome ?nome) (valor ?valor))
 =>
 (if (eq (nth$ 1 ?rest) and)
 then (modify ?f (precursors (rest$ ?rest)))
 else (modify ?f (precursors ?rest))))

(defrule PERGUNTAS::precursor-is-not-satisfied
 ?f <- (pergunta (already-asked FALSE)
 (precursors ?nome is-not ?valor $?rest))
 (atributos (nome ?nome) (valor ~?valor))
 =>
 (if (eq (nth$ 1 ?rest) and)
 then (modify ?f (precursors (rest$ ?rest)))
 else (modify ?f (precursors ?rest))))
```

Importaremos os dados do módulo perguntas para que sejam aplicados durante a criação
das perguntas e seu respectivo banco

```
(defmodule vinho-PERGUNTAS (import PERGUNTAS ? ALL))
```

```
(deffacts vinho-PERGUNTAS::pergunta-caracteristicas
 (pergunta (atributos componente-principal)
          (questao "O componente principal do prato é carne, peixe, ou frango? ")
                   (respostas-validas carne peixe frango desconhecido))
 (pergunta (atributos tem-farora)
         (precursors componente-principal is frango)
         (questao "A comida vem com farofa? ")
         (respostas-validas sim nao desconhecido))
 (pergunta (atributos tem-farora)
         (precursors componente-principal is carne)
         (questao "A comida vem com farofa? ")
         (respostas-validas sim nao desconhecido))
 (pergunta (atributos tem-farora)
         (precursors componente-principal is peixe)
         (questao "A comida vem com farofa? ")
         (respostas-validas sim nao desconhecido))
 (pergunta (atributos tem-farora)
                   (precursors componente-principal is desconhecido)
         (questao "A comida vem com farofa? ")
         (respostas-validas sim nao desconhecido))
 (pergunta (atributos tem-molho)
           (questao "A comida vem acompanhada de molho? ")
         (respostas-validas sim nao desconhecido))
 (pergunta (atributos molho)
         (precursors tem-molho is sim)
           (questao "O molho do prato é picante, doce, cremoso, ou tomate? ")
                   (respostas-validas molho picante doce cremoso tomate desconhecido))
 (pergunta (atributos sabor)
```

```
(questao "O sabor da comida é delicado, moderado, ou forte? ")
 (respostas-validas delicado moderado forte desconhecido))
 (pergunta (atributos preferencia-corpo)
 (questao "Você prefere vinho leve, medio, ou encorpado? ")
 (respostas-validas leve medio encorpado desconhecido))
 (pergunta (atributos preferencia-cor)
 (questao "você prefere vinho tinto ou branco? ")
 (respostas-validas tinto branco desconhecido))
 (pergunta (atributos preferencia-doçura)
 (questao "Você prefere vinho seco, medio, ou doce? ")
 (respostas-validas seco medio doce desconhecido)))
```

Criaremos o modulo de regras o qual importará dados relativos a porcentagem:

```
(defmodule REGRAS (import MAIN ?ALL) (export ?ALL))
```

Declaração das variáveis do módulo:

```
(deftemplate REGRAS::rule
 (slot porcentagem (default 100.0))
 (multislot if)
 (multislot then))
```

Criação das regras relativas às perguntas:

```
(defrule REGRAS::joga-fora-anterior
 ?f <- (rule (if and $?rest))
 =>
 (modify ?f (if ?rest)))

(defrule REGRAS::joga-fora-posterior
 ?f <- (rule (then and $?rest))
 =>
 (modify ?f (then ?rest)))
```

```
(defrule REGRAS::remove-condicao-verdade
 ?f <- (rule (porcentagem ?c1)
 (if ?atributos is ?valor $?rest))
 (atributos (nome ?atributos)
 (valor ?valor)
 (porcentagem ?c2))
=>
 (modify ?f (porcentagem (min ?c1 ?c2)) (if ?rest)))

(defrule REGRAS::remove-condicao-falsa
 ?f <- (rule (porcentagem ?c1)
          (if ?atributos is-not ?valor $?rest))
 (atributos (nome ?atributos) (valor ~?valor) (porcentagem ?
c2))
 =>
 (modify ?f (porcentagem (min ?c1 ?c2)) (if ?rest)))

(defrule                REGRAS::perform-rule-consequent-com-
porcentagem
 ?f <- (rule (porcentagem ?c1)
          (if)
             (then ?atributos is ?valor com porcentagem ?c2 $?
rest))
 =>
 (modify ?f (then ?rest))
 (assert (atributos (nome ?atributos)
                 (valor ?valor)
                 (porcentagem (/ (* ?c1 ?c2) 100)))))

(defrule            REGRAS::perform-rule-consequent-comout-
porcentagem
 ?f <- (rule (porcentagem ?c1)
          (if)
          (then ?atributos is ?valor $?rest))
 (test (or (eq (length$ ?rest) 0)
         (neq (nth$ 1 ?rest) com)))
```

```
=>
 (modify ?f (then ?rest))
  (assert (atributos (nome ?atributos) (valor ?valor) (porcentagem ?c1))))
```

Criaremos o modulo de seleção do melhor vinho possível:

```
(defmodule ESCOLHA-QUALIDADES (import REGRAS ? ALL)
                    (import PERGUNTAS ?ALL)
                    (import MAIN ?ALL))

(defrule ESCOLHA-QUALIDADES::startit => (focus REGRAS))
```

Criação das regras para escolha das melhores características de vinho:

```
(deffacts the-vinho-rules
 (rule (if tem-molho is sim and
        molho is picante)
     (then melhor-corpo is encorpado))

 (rule (if sabor is delicado)
     (then melhor-corpo is leve))

 (rule (if sabor is average)
     (then melhor-corpo is leve com porcentagem 30 and
         melhor-corpo is medio com porcentagem 60 and
             melhor-corpo is encorpado com porcentagem 30))

 (rule (if sabor is forte)
       (then melhor-corpo is medio com porcentagem 40 and
             melhor-corpo is encorpado com porcentagem 80))
```

```
(rule (if tem-molho is sim and
 molho is cremoso)
 (then melhor-corpo is medio com porcentagem 40 and
 melhor-corpo is encorpado com porcentagem 60))

 (rule (if preferencia-corpo is encorpado)
 (then melhor-corpo is encorpado com porcentagem 40))

 (rule (if preferencia-corpo is medio)
 (then melhor-corpo is medio com porcentagem 40))

 (rule (if preferencia-corpo is leve)
 (then melhor-corpo is leve com porcentagem 40))

 (rule (if preferencia-corpo is leve and
 melhor-corpo is encorpado)
 (then melhor-corpo is medio))

 (rule (if preferencia-corpo is encorpado and
 melhor-corpo is leve)
 (then melhor-corpo is medio))

 (rule (if preferencia-corpo is desconhecido)
 (then melhor-corpo is leve com porcentagem 20 and
 melhor-corpo is medio com porcentagem 20 and
 melhor-corpo is encorpado com porcentagem 20))
```

Criação das regras para escolha da melhor cor de vinho:

```
 (rule (if componente-principal is carne)
      (then melhor-cor is tinto com porcentagem 90))

 (rule (if componente-principal is carne and
           tem-farora is sim)
      (then melhor-cor is tinto com porcentagem 80 and
           melhor-cor is branco com porcentagem 50))
```

```
(rule (if tem-molho is sim and
  (rule (if componente-principal is carne and
 tem-farora is nao)
 (then melhor-cor is branco com porcentagem 70 and
 melhor-cor is tinto com porcentagem 60))

 (rule (if componente-principal is frango and
 tem-farora is nao)
 (then melhor-cor is branco com porcentagem 90 and
 melhor-cor is tinto com porcentagem 30))

(rule (if componente-principal is frango and
        tem-farora is sim)
     (then melhor-cor is tinto com porcentagem 80 and
          melhor-cor is branco com porcentagem 50))

 (rule (if componente-principal is peixe)
     (then melhor-cor is branco))

 (rule (if componente-principal is peixe and
        tem-farora is sim)
     (then melhor-cor is branco com porcentagem 80 and
          melhor-cor is tinto com porcentagem 50))

 (rule (if componente-principal is peixe and
        tem-farora is nao)
     (then melhor-cor is branco com porcentagem 70 and
          melhor-cor is tinto com porcentagem 40))

 (rule (if componente-principal is-not peixe and
        tem-molho is sim and
        molho is tomate)
     (then melhor-cor is tinto))

 (rule (if tem-molho is sim and
        molho is cremoso)
     (then melhor-cor is branco com porcentagem 40))
```

```
(rule (if preferencia-cor is tinto)
 (then melhor-cor is tinto com porcentagem 40))
 (rule (if preferencia-cor is branco)
      (then melhor-cor is branco com porcentagem 40))

 (rule (if preferencia-cor is desconhecido)
      (then melhor-cor is tinto com porcentagem 20 and
           melhor-cor is branco com porcentagem 20))

 (rule (if componente-principal is desconhecido and
          tem-farora is sim)
      (then melhor-cor is branco com porcentagem 30 and
           melhor-cor is tinto com porcentagem 20))

  (rule (if componente-principal is desconhecido and
          tem-farora is nao)
      (then melhor-cor is tinto com porcentagem 10 and
           melhor-cor is branco com porcentagem 20))

 (rule (if componente-principal is desconhecido and
          tem-molho is nao)
      (then melhor-cor is tinto com porcentagem 20 and
           melhor-cor is while com porcentagem 30))

 (rule (if componente-principal is desconhecido and
          tem-molho is sim)
      (then melhor-cor is branco com porcentagem 30 and
           melhor-cor is tinto com porcentagem 20))
```

Criação das regras para escolha da doçura do vinho:

```
 (rule (if tem-molho is sim and
          molho is doce)
      (then melhor-doçura is doce com porcentagem 90 and
           melhor-doçura is medio com porcentagem 40))
```

```
(rule (if preferencia-doçura is seco)
 (then melhor-doçura is seco com porcentagem 40))

 (rule (if preferencia-doçura is medio)
 (then melhor-doçura is medio com porcentagem 40))

 (rule (if preferencia-doçura is doce)
 (then melhor-doçura is doce com porcentagem 40))

 (rule (if melhor-doçura is doce and
 preferencia-doçura is seco)
 (then melhor-doçura is medio))

 (rule (if melhor-doçura is seco and
 preferencia-doçura is doce)
 (then melhor-doçura is medio))

 (rule (if preferencia-doçura is desconhecido)
      (then melhor-doçura is seco com porcentagem 20 and
           melhor-doçura is medio com porcentagem 20 and
           melhor-doçura is doce com porcentagem 20))
```

Definição do módulo que conterá as informações dos vinhos:

```
(defmodule VINHOS (import MAIN ?ALL))
```

Definição do banco dos vinhos e declaração de variáveis:

```
(deffacts qualquer-caracteristicas
 (atributos (nome melhor-cor) (valor any))
 (atributos (nome melhor-corpo) (valor any))
 (atributos (nome melhor-doçura) (valor any)))
```

```
deftemplate VINHOS::vinho
 (slot nome (default ?NONE))
 (multislot cor (default any))
 (multislot corpo (default any))
 (multislot doçura (default any)))
```

Banco de dados dos vinhos e suas características:

```
(deffacts VINHOS::the-vinho-list
  (vinho (nome Gamay) (cor tinto) (corpo medio) (doçura medio doce))
  (vinho (nome Chablis) (cor branco) (corpo leve) (doçura seco))
 (vinho (nome Sauvignon-Blanc) (cor branco) (corpo medio) (doçura seco))
  (vinho (nome Chardonnay) (cor branco) (corpo medio encorpado) (doçura medio seco))
  (vinho (nome Tempranilo) (cor tinto) (corpo encorpado) (doçura seco))
  (vinho (nome Riesling) (cor branco) (corpo leve medio) (doçura medio doce))
   (vinho (nome Geverztraminer) (cor branco) (corpo encorpado))
  (vinho (nome Chenin-Blanc) (cor branco) (corpo leve) (doçura medio doce))
 (vinho (nome Valpolicella) (cor tinto) (corpo leve))
 (vinho (nome Cabernet-Sauvignon) (cor tinto) (doçura seco medio))
 (vinho (nome Zinfandel) (cor tinto) (doçura seco medio))
 (vinho (nome Pinot-noir) (cor tinto) (corpo medio) (doçura medio))
 (vinho (nome Burgundy) (cor tinto) (corpo encorpado)))
```

Regra que gera as porcentagens dos vinhos baseados nas escolhas:

```
(defrule VINHOS::generate-vinhos
 (vinho (nome ?nome)
      (cor $? ?c $?)
      (corpo $? ?b $?)
      (doçura $? ?s $?))
 (atributos (nome melhor-cor) (valor ?c) (porcentagem ?
porcentagem-1))
 (atributos (nome melhor-corpo) (valor ?b) (porcentagem
?porcentagem-2))
 (atributos (nome melhor-doçura) (valor ?s) (porcentagem ?
porcentagem-3))
 =>
 (assert (atributos (nome vinho) (valor ?nome)
                    (porcentagem (min ?porcentagem-1 ?
porcentagem-2 ?porcentagem-3)))))
```

Criaremos o módulo que mostrará os resultados:

```
(defmodule MOSTRA-RESULTADO (import MAIN ?ALL))
```

Definiremos a regra que mostrará os resultados:

```
(defrule MOSTRA-RESULTADO::header ""
  (declare (salience 10))
  =>
  (println)
  (println "      Vinho Escolhido" crlf)
  (println " Vinho              PORCENTAGEM")
  (println " ------------------------------")
  (assert (phase print-VINHOS)))
```

```
(defrule MOSTRA-RESULTADO::print-vinho ""
   ?rem <- (atributos (nome vinho) (valor ?nome)
(porcentagem ?per))
 (not (atributos (nome vinho) (porcentagem ?per1&:(> ?
per1 ?per))))

=>
 (retract ?rem)
 (format t " %-24s %2d%%%n" ?nome ?per))

(defrule  MOSTRA-RESULTADO::remove-porcentagens-
baixas ""
 ?rem <- (atributos (nome vinho) (porcentagem ?per&:(<
?per 20)))
 =>
 (retract ?rem))

(defrule MOSTRA-RESULTADO::end-spaces ""
  (not (atributos (nome vinho)))
  =>
  (println))
```

3.CONSIDERAÇÕES FINAIS

Inicialmente, o foco de nosso trabalho era outro, porém, ao pesquisar e entrar em contato com os conteúdos disponíveis sobre a plataforma CLIPS, percebeu-se que a maior parte do material disponível para aprendermos sobre a linguagem estava na língua inglesa, o que dificultou a compreensão para execução dos processos iniciais do projeto. Então, pensando nos futuros estudantes do CLIPS, resolvemos desenvolver um tutorial simples sobre a plataforma com o intuito de ensinar uma nova forma de programar, mesmo a mesma sendo uma plataforma antiga. Mas, ainda assim, não deixamos de lado a nossa ideia inicial de fazer um sistema especialista usando a Inteligência Artificial.

Ao longo da execução, desenvolvemos três cases e exemplos de sistemas especialista, os quais são apresentados no trabalho. O primeiro, case 1, sendo o básico e tradicional “hello world", utilizando a inteligência artificial. No case 2 foi desenvolvido um sistema especialista capaz de dar possíveis diagnósticos a partir da temperatura informada pelo usuário. E por fim, com o case 3, há um desenvolvimento mais complexo e o mais semelhante a um SE, um sistema que, a partir das escolhas do usuário, defini e responde com o melhor vinho para o prato e os alimentos escolhidos pelo mesmo.

Finalizamos esse trabalho com um grande sentimento de dever cumprido e instigando o leitor a conhecer um pouco mais sobre Inteligência Artificial, Sistemas Especialistas e obviamente, a plataforma CLIPS. Não menos importante, esse trabalho é um meio para incentivar a educação, essa que é essencial para a formação da próxima geração de estudantes do IFSP.

4. REFERÊNCIAS BIBLIOGRÁFICAS

https://en.wikipedia.org/wiki/CLIPS
(Acessado em 22/08/2022)
http://campeche.inf.furb.br/tccs/2002-I/2002-1ronaldocesarschorkjuniorvf.pdf
(Acessado em 22/08/2022)
https://www.cin.ufpe.br/~frcc/meus_arquivos/das6607-francesca-maiara-ricardo
(Acessado em 22/08/2022)
https://clipsrules.sourceforge.io/documentation/v630/bpg.pdf
(Acessado em 02/09/2022)
https://www.hpe.com/br/pt/what-is/artificial-intelligence.html#:~:text=A%20intelig%C3%AAncia%20artificial%20%C3%A9%20um,para%20demonstrar%20comportamento%20%E2%80%9Cinteligente%E2%80%9D
(Acessado em 02/09/2022)
https://www.totvs.com/blog/inovacoes/o-que-e-inteligencia-artificial/
(Acessado em 02/09/2022)
https://www.oracle.com/br/artificial-intelligence/what-is-ai/
(Acessado em 02/09/2022)
https://fia.com.br/blog/inteligencia-artificial/
(Acessado em 02/09/2022)
https://tecnoblog.net/responde/o-que-e-inteligencia-artificial/
(Acessado em 02/09/2022)