

Tema1 - Projeto de Compiladores:

“Desenvolvimento de um Compilador para a Linguagem MiniPar versão 2025.1”

Implementação de Analisadores (Léxico - Lexer, Sintático – Parser - Arvore Sintática AST - Semântico), Tabela de Símbolos, , Geração de Código Intermediário (3 endereços), Geração de Código Assembly ARMv7 e Execução desse Código Assembly no CPULATOR - emulador na web)

respeitando a Gramática da Linguagem do Minipar 2025.1 e os programas de Teste 2025.1

Projeto do Compilador MiniPar 2025.1 baseado no Reuso de Software utilizando Componentes de Software

1- ENUNCIADO:

Desenvolvimento de um Compilador para a Linguagem MiniPar utilizando Componentes de Software” (Analisadores Léxico, Sintático, Semântico, Tabela de Símbolos, Gerador de Código de Três Endereços e Gerador de Código Assembly ARMv7).

Execução do Compilador segundo a Gramática da Linguagem Minipar versão 2025.1.

Implementar o Compilador da linguagem MiniPar 2025.1 em Java ou Phyton ou C++.

A modelagem do Compilador para a linguagem MiniPar deve ser realizada utilizando técnicas de Engenharia de Software tais como Componentes de Software, a arquitetura do Compilador será realizada utilizando Componentes de Software, os Analisadores (Léxico – Sintático - Semântico) e a Tabela de Símbolos serão representados via componentes de software. Também os Geradores (Código Intermediário de Três Endereços e Código Assembly) serão representados via Componentes de Software. No contexto da modelagem dos Componentes de Software deve ser utilizada a UML (Linguagem Unificada de Modelagem de Sistemas Orientados a Objetos).

2 - A LINGUAGEM MINIPAR 2025.1

Programa_MiniPar: Programa de instruções de execução sequencial e paralela.

Características

Execução de PAR precisa da utilização de Threads para implementar o paralelismo. Considerar a execução de Threads como Processos independentes sem compartilhar informações.

Comunicação somente via mensagem.

Canal de Comunicação (implementado com sockets em Java ou Phyton)

O Send e Receive de dados via un canal de comunicação que pode ser implementado via sockets em Java ou Phyton.

Send (envio de dados via um canal de comunicação - socket a outro computador)

Receive (recepção de dados de outro computador via um canal de comunicação - socket)

Os testes podem ser realizados no mesmo computador na forma localhost.

Blocos de Instruções SEQ e PAR

SEQ - Execução sequencial (tradicional) de um conjunto de instruções

PAR - Execução paralela (implementada com Threads) de um conjunto de instruções

Tipos de Variáveis

Bool – o mesmo do Phyton

Int - o mesmo do Phyton

String - o mesmo do Phyton

c_channel – (comunication channel) canal de comunicação entre dois computadores

Entrada e Saída

Input (entrada do Teclado) – a mesma do Phyton

Output (na tela) – a mesma do Phyton

Uso de Comentários #

Com declaração e uso de Funções

Observação recuperar a Gramatica do MINIPAR 2024.2 e

Concluir as Produções e Acrescentar novas Produções à Gramática BNF considerando obrigatoriamente os Programas de Teste 2025.1

Produções

programa_minipar → bloco_stmt
bloco_stmt → bloco_SEQ | bloco_PAR
bloco_SEQ → SEQ stmts
bloco_PAR → PAR stmts
tipos_var → ...
stmts → atribuição | if (bool) stmt | if (bool) stmt else stmt | while (bool) stmt
atribuição → id = expr
expr → ...
c_channel → chan id id_comp1 id_comp2
...

Observação: No contexto do Minipar 2025.1 deve-se recuperar e atualizar (considerando os Programas de Teste do período 2025.1) a Gramática do Minipar 2024.2 acrescentando os Geradores (Código Intermediário de Três Endereços e Código Assembly ARMv7) solicitados

Observação 1:

programa_minipar – Representa um programa a ser executado pelo Compilador MiniPar

Observação 2: Deve se considerar precedência nos operadores aritméticos +, -,* e / no momento de completar a produção expr, para isto revisar o material da disciplina Tradução Dirigida por Sintaxe.

Observação 3: Deve se utilizar funções e recursão.

Observação 4: Precisam concluir a Gramática da Linguagem MINIPAR para que cumpra com a especificação de todos os programas de testes 2025.1.

Observação 5: Realizar Tratamento de Erros na Implementação do Compilador MiniPar 2025.1.

Programas de Teste (na linguagem MINIPAR 2025.1)

Lembrar de utilizar comentários via #

Programa de Teste 1.

```
programa-miniPar

# Programa cliente servidor de uma calculadora aritmética simples

# O cliente (computador_1) solicita a execução de uma operação aritmética e
# o servidor (computador_2) realiza o
# cálculo retornando o resultado para o cliente

c_channel calculadora computador_1 computador_2

# declaração do canal de comunicação calculadora associada a dos computadores
# (computador_1 e computador_2).
```

SEQ

```
# depois do SEQ todas as seguintes instruções indentadas serão executadas de
# forma seqüencial

# Apresentar na tela via print as opções da calculadora +, -, *, /

# Ler a operação aritmética desejada via sys.stdin.readline()

# Ler o operando 1 (valor) e operando 2 (valor) via sys.stdin.readline()
```

```
calculadora.send (operação, valor1, valor2, resultado)

# computador cliente (computador_1)

# Imprime o resultado via print
```

---- Execução: Servidor (computador 2)

```
# computador servidor

calculadora.receive (operação, valor1, valor2, resultado)

# computador 2 recebe a solicitação do cliente (computador 1) e executa o cálculo e
# retorna o resultado ao cliente (computador 1)
```

Execução: Testar com localhost para o computador 2 (Servidor)

Programa de Teste 2.

program-minipar

PAR

```
# Execução paralela (implementação via uso de threads em Java ou
# Phyton)
# depois do PAR todas as seguintes instruções indentadas serão executadas de
# forma “paralela”
```

```
# A seguir colocar o Código do Cálculo do Fatorial de um número (thread 1)
... linhas de código do Cálculo do Fatorial de um número na linguagem
    MiniPar
```

```
# A seguir colocar o Código do Cálculo da Série de Fibonacci (thread 2)
... linhas de código do Cálculo da Série de Fibonacci na linguagem
    MiniPar
```

Execução: Thread 1 (Cálculo do Fatorial) e thread 2 (Cálculo da Série de Fibonacci) são executadas de forma simultânea (“paralela”) no mesmo computador.

Dica:

http://www.dicas-l.com.br/arquivo/programando_socket_em_c++_sem_secreto.php

Programa de Teste 3.

#implementar um programa na linguagem minipar para o código seguinte
 #(python) da implementação de um neurônio

```
# Código Simples de um Neuronio

input_val = 1
output_desire = 0

input_weight = 0.5
learning_rate = 0.01

# Função de Ativação
def activation(sum):
    if sum >= 0:
        return 1
    else:
        return 0

print("Entrada: ", input_val, " Desejado: ", output_desire)

# Inicializar erro
error = 1000.0
iteration = 0
bias = 1
bias_weight = 0.5

while error != 0:
    iteration += 1
    print("#### Iteração: ", iteration)
    print("Peso: ", input_weight)

    sum_val = (input_val * input_weight) + (bias * bias_weight)

    output = activation(sum_val)
    print("Saída: ", output)

    error = output_desire - output
    print("Erro: ", error)

    if error != 0:
        input_weight = input_weight + (learning_rate * input_val * error)
        print("Peso do bias: ", bias_weight)
        bias_weight = bias_weight + (learning_rate * bias * error)

print("Parabéns!!! A Rede de um Neurônio Aprendeu")
print ("Valor desejadao: ", output_desire)
```

Execução (parte da execução ja que são muitas iterações):

Entrada: 1 Desejado: 0

Iteração: 1

Peso: 0.5

Saída: 1

Erro: -1

Peso do bias: 0.5

Iteração: 2

Peso: 0.49

Saída: 1

Erro: -1

Peso do bias: 0.49

Iteração: 3

Peso: 0.48

Saída: 1

Erro: -1

Peso do bias: 0.48

Iteração: 4

Peso: 0.47

Saída: 1

Erro: -1

Peso do bias: 0.47

...

Iteração: 49

Peso: 0.0200

Saída: 1

Erro: -1

Peso do bias: 0.0200

Iteração: 50

Peso: 0.0100

Saída: 1

Erro: -1

Peso do bias: 0.0100

Iteração: 51

Peso: -0.0000

Saída: 0

Erro: 0

Parabéns! O neurônio aprendeu.

Valor desejado: 0

...Program finished with exit code 0

Press ENTER to exit console.

Programa de Teste 4.

#implementar um programa na linguagem Minipar para o código seguinte
#(python) da implementação de um a Rede Neural para aprender a função
#XOR com uma camada oculta de #3 neurônios e uma saída. Utilizando
#funções.

#Somente usando funções da Linguagem MINIPAR

```
#Este código cria uma rede neural com uma camada oculta de três
neurônios
#e uma camada de saída com um neurônio, utilizando a função de
ativação sigmóide.
#Ele treina a rede para aprender a função XOR usando feedforward e
backpropagation.
#Todos os cálculos são realizados manualmente, sem o uso de
bibliotecas externas.
```

```
import random
import math

# Funções auxiliares

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

class Neuronio:
    def __init__(self, num_inputs):
        self.pesos = [random.random() for _ in range(num_inputs)]
        self.bias = random.random()
        self.saida = 0

    def feedforward(self, entradas):
        soma = sum([entradas[i] * self.pesos[i] for i in range(len(entradas))]) + self.bias
        self.saida = sigmoid(soma)
        return self.saida

    def calcular_derivada(self):
        return sigmoid_derivative(self.saida)

class RedeNeural:
    def __init__(self, taxa_aprendizado=0.2):
        self.taxa_aprendizado = taxa_aprendizado
        self.camada_oculta = [Neuronio(2) for _ in range(3)]
        self.neuronio_saida = Neuronio(3)
```

```

def feedforward(self, entradas):
    saidas_ocultas = [neuronio.feedforward(entradas) for neuronio in self.camada_oculta]
    saida_final = self.neuronio_saida.feedforward(saidas_ocultas)
    return saidas_ocultas, saida_final

def backpropagation(self, entradas, saida_desejada, saidas_ocultas, saida_final):
    erro = saida_desejada - saida_final
    delta_saida = erro * self.neuronio_saida.calcular_derivada()

    # Atualiza pesos da camada de saída
    for i in range(len(self.neuronio_saida.pesos)):
        self.neuronio_saida.pesos[i] += saidas_ocultas[i] * delta_saida * self.taxa_aprendizado
        self.neuronio_saida.bias += delta_saida * self.taxa_aprendizado

    # Atualiza pesos da camada oculta
    for i, neuronio in enumerate(self.camada_oculta):
        delta_oculto = delta_saida * self.neuronio_saida.pesos[i] * neuronio.calcular_derivada()
        for j in range(len(neuronio.pesos)):
            neuronio.pesos[j] += entradas[j] * delta_oculto * self.taxa_aprendizado
            neuronio.bias += delta_oculto * self.taxa_aprendizado

def treinar(self, entradas, saidas_desejadas, epochas=20000):
    for _ in range(epochas):
        for entrada, saida_desejada in zip(entradas, saidas_desejadas):
            saidas_ocultas, saida_final = self.feedforward(entrada)
            self.backpropagation(entrada, saida_desejada, saidas_ocultas, saida_final)

def testar(self, entradas):
    for entrada in entradas:

```

```
_ , saida_final = self.feedforward(entrada)
print(f"Input: {entrada}, Predicted Output: {saida_final:.4f}")

# Dados XOR
entradas = [[0, 0], [0, 1], [1, 0], [1, 1]]
saidas_desejadas = [0, 1, 1, 0]

# Execução
rede = RedeNeural()
rede.treinar(entradas, saidas_desejadas)
rede.testar(entradas)
```

Execução:

```
Input: [0, 0], Predicted Output: 0.0089
Input: [0, 1], Predicted Output: 0.9769
Input: [1, 0], Predicted Output: 0.9758
Input: [1, 1], Predicted Output: 0.0291
```

...Program finished with exit code 0

Press ENTER to exit console.

Programa de Teste 5.

#Implementar um programa na linguagem Minipar para o código seguinte #(python) da implementação de um Sistema de Recomendação, no contexto de #e-commerce levando em conta o histórico de compras do usuário, utilizando #Redes Neurais e implementado em Python.

#Somente usando funções da Linguagem MINIPAR

Estrutura da Rede Neural do Programa

1. Entrada

Cada produto é representado por um número:

1 se o usuário comprou

0 se não comprou

Isso forma um vetor binário chamado histórico codificado.

2. Camada Oculta

Tem 10 neurônios.

Cada neurônio recebe todos os valores da entrada, multiplica por pesos e soma com um bias.

A soma passa por uma função de ativação ReLU:

python

relu(x) = max(0, x)

Isso ajuda a introduzir não-linearidade, ignorando valores negativos.

3. Camada de Saída

Tem o mesmo número de neurônios que a entrada (um para cada produto).

Cada neurônio recebe os valores da camada oculta, aplica pesos e bias, e passa por a função sigmóide:

python

sigmoid(x) = 1 / (1 + e^{-x})

A saída será um valor entre 0 e 1, que representa a probabilidade de recomendação.

→ Propagação (Forward Propagation)

Esse é o processo de passar os dados pela rede:

Multiplica os dados de entrada pelos pesos da camada oculta.

Soma com os bias e aplica ReLU.

Usa os resultados da camada oculta como entrada para a camada de saída.

Aplica a função sigmóide para obter as probabilidades finais.

```
import math

class Produto:
    def __init__(self, nome):
        self.nome = nome

class Categoria:
    def __init__(self, nome, produtos):
        self.nome = nome
        self.produtos = [Produto(p) for p in produtos]

class Usuario:
    def __init__(self, historico_compras):
        self.historico_compras = historico_compras

    def codificar_historico(self, todos_produtos):
        codificacao = [1 if produto in self.historico_compras else 0 for produto in todos_produtos]
        return codificacao

class RedeNeural:
    def __init__(self, input_size, hidden_size, output_size):
        self.W1 = [[0.5 for _ in range(hidden_size)] for _ in range(input_size)]
        self.b1 = [0.5] * hidden_size
        self.W2 = [[0.5 for _ in range(output_size)] for _ in range(hidden_size)]
        self.b2 = [0.5] * output_size

    def relu(self, x):
        return [max(0, i) for i in x]
```

```

def sigmoid(self, x):
    return [1 / (1 + math.exp(-i)) for i in x]

def forward(self, X):
    Z1 = [sum(X[j] * self.W1[j][i] for j in range(len(X))) + self.b1[i] for i in
range(len(self.b1))]
    A1 = self.relu(Z1)
    Z2 = [sum(A1[j] * self.W2[j][i] for j in range(len(A1))) + self.b2[i] for i in
range(len(self.b2))]
    A2 = self.sigmoid(Z2)
    return A2

class Recomendador:
    def __init__(self, usuario, categorias):
        self.usuario = usuario
        self.categorias = categorias
        self.todos_produtos = [produto.nome for categoria in categorias for produto in
categoria.produtos]

    def recomendar(self):
        entrada_codificada = self.usuario.codificar_historico(self.todos_produtos)
        input_size = len(entrada_codificada)
        hidden_size = 10
        output_size = len(entrada_codificada)

        rede = RedeNeural(input_size, hidden_size, output_size)
        saida = rede.forward(entrada_codificada)

        recomendacoes = [
            self.todos_produtos[i]
            for i in range(len(saida))
        ]

```

```

        if saida[i] > 0.5 and self.todos_produtos[i] not in self.usuario.historico_compras
    ]
    return recomendacoes

# ----- Execução -----

# Dados de entrada
categorias = [
    Categoria("Eletrônicos", ["Smartphone", "Laptop", "Tablet", "Fones de ouvido"]),
    Categoria("Roupas", ["Camisa", "Jeans", "Jaqueta", "Sapatos"]),
    Categoria("Eletrodomésticos", ["Geladeira", "Micro-ondas", "Máquina de lavar", "Ar condicionado"]),
    Categoria("Livros", ["Ficção", "Não-ficção", "Ficção científica", "Fantasia"])
]

usuario = Usuario(["Smartphone", "Jeans", "Micro-ondas", "Ficção"])
recomendador = Recomendador(usuario, categorias)
recomendacoes = recomendador.recomendar()

# Exibir recomendações
print("Produtos recomendados para você:")
for produto in recomendacoes:
    print(produto)

```

Execução:

```

Produtos recomendados para você:
Laptop
Tablet
Fones de ouvido
Camisa
Jaqueta
Sapatos
Geladeira
Máquina de lavar
Ar condicionado
Não-ficção

```

```

Ficão científica
Fantasia
...Program finished with exit code 0
Press ENTER to exit console.

```

Programa de Teste 6.

#Implementar um programa na linguagem Minipar para o código seguinte
(python) da implementação do fatorial de 5

#Somente usando funções da Linguagem MINIPAR

```

# Cálculo do fatorial de 5 de forma iterativa
numero = 5
fatorial = 1

for i in range(1, numero + 1):
    fatorial *= i

print("O fatorial de", numero, "é", fatorial)

```

Execução:

```

O fatorial de 5 é 120
...Program finished with exit code 0
Press ENTER to exit console.

```

Código gerado (Assembly ARMv7) a rodar no Emulador CPULATOR
- Rodou OK

```

.global _start

_start:
    MOV     R0, #1          @ fatorial = 1
    MOV     R1, #1          @ i = 1
    MOV     R2, #5          @ limite = 5

loop:
    CMP     R1, R2          @ compara i com 5
    BGT     end             @ se i > 5, sai do loop

    MUL     R0, R0, R1      @ fatorial *= i
    ADD     R1, R1, #1      @ i += 1
    B      loop            @ repete o loop

```

```

end:
@ Resultado final está em R0
@ Para visualizar no CPULATOR, pode armazenar em memória

LDR      R3, =RESULT    @ endereço de armazenamento
STR      R0, [R3]        @ salva fatorial em RESULT

B       .                @ fim do programa (loop infinito)

.data
RESULT: .word 0           @ espaço reservado para o resultado

```

Programa de Teste 7.

#Implementar um programa na linguagem Minipar para o código seguinte
#(python) da implementação do serie de Fibonacci com 5 termos

#Somente usando funções da Linguagem MINIPAR

Série de Fibonacci com 5 termos

n = 5

a = 0

b = 1

print("Série de Fibonacci com", n, "termos:")

for i in range(n):

print(a)

proximo = a + b

a = b

b = próximo

Execução-OK

Série de Fibonacci com 5 termos:

0

1

1

2

3

...Program finished with exit code 0

Press ENTER to exit console.

Código gerado (Assembly ARMv7) a rodar no Emulador CPULATOR - Rodou OK

```
.global _start

_start:
    MOV     R0, #0          @ a = 0
    MOV     R1, #1          @ b = 1
    MOV     R2, #0          @ contador = 0
    MOV     R3, #5          @ número de termos = 5
    LDR     R4, =FIB_MEM   @ ponteiro para memória de saída

loop:
    CMP     R2, R3          @ verifica se contador < 5
    BGE     end              @ se contador >= 5, termina

    STR     R0, [R4], #4      @ armazena 'a' na memória e incrementa
    ponteiro

    ADD     R5, R0, R1        @ proximo = a + b
    MOV     R0, R1          @ a = b
    MOV     R1, R5          @ b = proximo

    ADD     R2, R2, #1        @ contador += 1
    B      loop

end:
    B      .                  @ loop infinito para encerrar

.data
FIB_MEM: .space 20          @ espaço para 5 inteiros (5 x 4 bytes)
```

Programa de Teste 8.

#Implementar um programa na linguagem Minipar para o código seguinte
#(python) da implementação do Quicksort não recursivo de um array contendo
#5 elementos a serem ordenados

#Somente usando funções da Linguagem MINIPAR

```
# Quicksort não recursivo tradicional
def quicksort_iterativo(arr):
    # Pilha com tuplas (início, fim)
    stack = [(0, len(arr) - 1)]
```

```

while stack:
    start, end = stack.pop()

    if start >= end:
        continue

    # Particionamento
    pivot = arr[end]
    i = start - 1

    for j in range(start, end):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    i += 1
    arr[i], arr[end] = arr[end], arr[i]

    # Empilha subarrays à esquerda e à direita do pivô
    stack.append((start, i - 1))
    stack.append((i + 1, end))

# Exemplo de ordenação com 5 elementos
array = [33, 12, 98, 5, 61]
quicksort_iterativo(array)
print("Array ordenado:", array)

```

Execução-OK

Array ordenado: [5, 12, 33, 61, 98]

...Program finished with exit code 0
Press ENTER to exit console.

Código Intermediário Gerado (Código de Três Endereços)

```

t0 = 0                      // início do array
t1 = 4                      // fim do array
S[0] = t0                   // empilha start
S[1] = t1                   // empilha end
sp = 2                      // topo da pilha (próxima posição livre)

LOOP:
    if sp == 0 goto END      // pilha vazia → fim

    sp = sp - 1

```

```

t2 = S[sp]           // end
sp = sp - 1
t3 = S[sp]           // start

if t3 >= t2 goto LOOP // intervalo inválido → ignora

t4 = A[t2]           // pivot = A[end]
t5 = t3 - 1          // i = start - 1
t6 = t3              // j = start

PARTITION_LOOP:
if t6 >= t2 goto PARTITION_END

t7 = A[t6]
if t7 <= t4 goto SWAP

t6 = t6 + 1
goto PARTITION_LOOP

SWAP:
t5 = t5 + 1
t8 = A[t5]
A[t5] = A[t6]
A[t6] = t8
t6 = t6 + 1
goto PARTITION_LOOP

PARTITION_END:
t5 = t5 + 1
t9 = A[t5]
A[t5] = A[t2]
A[t2] = t9

t10 = t5 - 1
S[sp] = t3           // empilha start
sp = sp + 1
S[sp] = t10          // empilha i - 1
sp = sp + 1

t11 = t5 + 1
S[sp] = t11          // empilha i + 1
sp = sp + 1
S[sp] = t2            // empilha end
sp = sp + 1

goto LOOP

END:
// Array A[0..4] está ordenado

```

Código gerado (Assembly ARMv7) a rodar no Emulador CPULATOR - Rodou OK

```
.global _start

.data
A:    .word 33, 12, 98, 5, 61      @ array a ser ordenado
S:    .space 40                  @ pilha com até 10 inteiros (5 pares)
sp:   .word 0                   @ ponteiro da pilha

.text
_start:
    LDR    R0, =S            @ R0 = endereço da pilha
    LDR    R1, =sp           @ R1 = endereço de sp
    MOV    R2, #0            @ start = 0
    MOV    R3, #4            @ end = 4

    STR    R2, [R0]          @ S[0] = start
    STR    R3, [R0, #4]       @ S[1] = end
    MOV    R4, #2            @ sp = 2
    STR    R4, [R1]          @ salva sp

LOOP:
    LDR    R4, [R1]          @ carrega sp
    CMP    R4, #0            @ se sp == 0, fim
    BEQ    END               @ se sp == 0, fim

    SUB    R4, R4, #1
    STR    R4, [R1]          @ sp -= 1
    LDR    R5, [R0, R4, LSL #2]  @ end = S[sp]

    SUB    R4, R4, #1
    STR    R4, [R1]          @ sp -= 1
    LDR    R6, [R0, R4, LSL #2]  @ start = S[sp]

    CMP    R6, R5
    BGE    LOOP              @ se start >= end, ignora

    LDR    R7, =A
    ADD    R8, R7, R5, LSL #2  @ R8 = &A[end]
    LDR    R9, [R8]           @ pivot = A[end]
    SUB    R10, R6, #1         @ i = start - 1
    MOV    R11, R6            @ j = start

PARTITION_LOOP:
```

```

    CMP  R11, R5
    BGE PARTITION_END

    ADD  R12, R7, R11, LSL #2 @ &A[j]
    LDR  R8, [R12]          @ A[j]
    CMP  R8, R9
    BLE SWAP

    ADD  R11, R11, #1
    B    PARTITION_LOOP

```

SWAP:

```

    ADD  R10, R10, #1      @ i++
    ADD  R2, R7, R10, LSL #2 @ &A[i]
    LDR  R3, [R2]          @ temp = A[i]
    STR  R8, [R2]          @ A[i] = A[j]
    STR  R3, [R12]         @ A[j] = temp
    ADD  R11, R11, #1
    B    PARTITION_LOOP

```

PARTITION_END:

```

    ADD  R10, R10, #1      @ i++
    ADD  R2, R7, R10, LSL #2 @ &A[i]
    ADD  R3, R7, R5, LSL #2 @ &A[end]
    LDR  R8, [R2]          @ temp = A[i]
    LDR  R9, [R3]          @ pivot = A[end]
    STR  R9, [R2]          @ A[i] = pivot
    STR  R8, [R3]          @ A[end] = temp

    LDR  R4, [R1]          @ sp
    ADD  R5, R10, #1       @ i + 1
    SUB  R6, R10, #1       @ i - 1

    STR  R6, [R0, R4, LSL #2] @ S[sp] = start
    ADD  R4, R4, #1
    STR  R4, [R1]
    STR  R10, [R0, R4, LSL #2] @ S[sp] = i - 1
    ADD  R4, R4, #1
    STR  R4, [R1]

    STR  R5, [R0, R4, LSL #2] @ S[sp] = i + 1
    ADD  R4, R4, #1
    STR  R4, [R1]
    STR  R5, [R0, R4, LSL #2] @ S[sp] = end
    ADD  R4, R4, #1
    STR  R4, [R1]

    B    LOOP

```

END:

B

No CPULator (Emulador) – Compilou e Rodou Ok

The screenshot shows the CPULator ARMv7 System Simulator interface. The top bar has tabs for Microsoft Copilot, Online Python Compiler, and the current window titled 'CPULator ARMv7 System Simulator'. The main window has a toolbar with buttons for Stopped, Step Into, Step Over, Step Out, Continue, Stop, Restart, Reload, File, and Help. On the left, there's a 'Registers' panel showing values for r2 through r10. The central area is the 'Editor (Ctrl-E)' which contains the following assembly code:

```
.global _start
.data
A: .word 33, 12, 98, 5, 61      @ array a ser ordenado
S: .space 40                      @ pilha com até 10 inteiros (5 pares)
sp: .word 0                        @ ponteiro da pilha

.text
_start:
    LDR R0, =S                  @ R0 = endereço da pilha
    LDR R1, =sp                 @ R1 = endereço de sp
    MOV R2, #0                  @ start = 0
    MOV R3, #4                  @ end = 4
```

Below the editor are tabs for Disassembly (Ctrl-D) and Memory (Ctrl-M). The bottom panel shows 'Messages' with the text 'Link: arm-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmvgoNJ.s.elf work/asmvgoNJ.s.o' and 'Compile succeeded.'.

O Resultado da Ordenação via Quicksort se encontra no ARRAY A que pode se consultado via a ABA Memory do CPULator.

3 – Colocar o pseudocódigo do Analisador Léxico (Lexer).

4 - Colocar o pseudocódigo do Analisador Sintático (Parser).

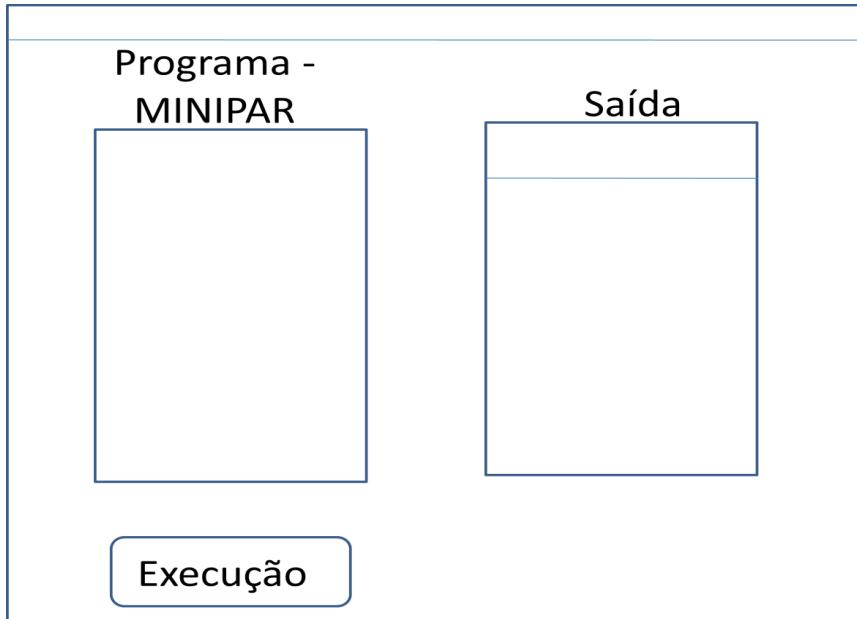
5 – Colocar o pseudocódigo do Analisador Semântico e o Tratamento de Erros, também deve permitir comentários via: #

6 – Colocar o pseudocódigo da Tabela de Símbolos.

7 – Colocar o pseudocódigo do Gerador de Código Intermediário de Três Endereços.

8 – Colocar o pseudocódigo do Gerador de Código Assembly ARMv7.

9 – Interface do Compilador da Linguagem MINIPAR



10 – Product Backlog do Compilador da Linguagem MINIPAR

11 – Sprints BackLog do Compilador da Linguagem MINIPAR

12 – Diagrama de Casos de Uso segundo a notação da UML

13 – Arquitetura utilizando Componentes de Software segundo a notação da UML

14 – Modelagem dos Componentes de Software via Diagrama de Classes segundo a notação da UML

15 – Testes de Software de Integração do Compilador da Linguagem MINIPAR

16 – Testes dos Programas de Exemplo na Linguagem MINIPAR

17 – Implementação

17.a Tecnologias Utilizadas (linguagem utilizada - qual foi o ambiente de desenvolvimento utilizado) na Implementação do Compilador da Linguagem Minipar. O código do Compilador Minipar implementado deve também conter comentários explicativos.

17.b Mostrar o passo a passo das telas da execução dos programas de Testes na Linguagem MINIPAR.

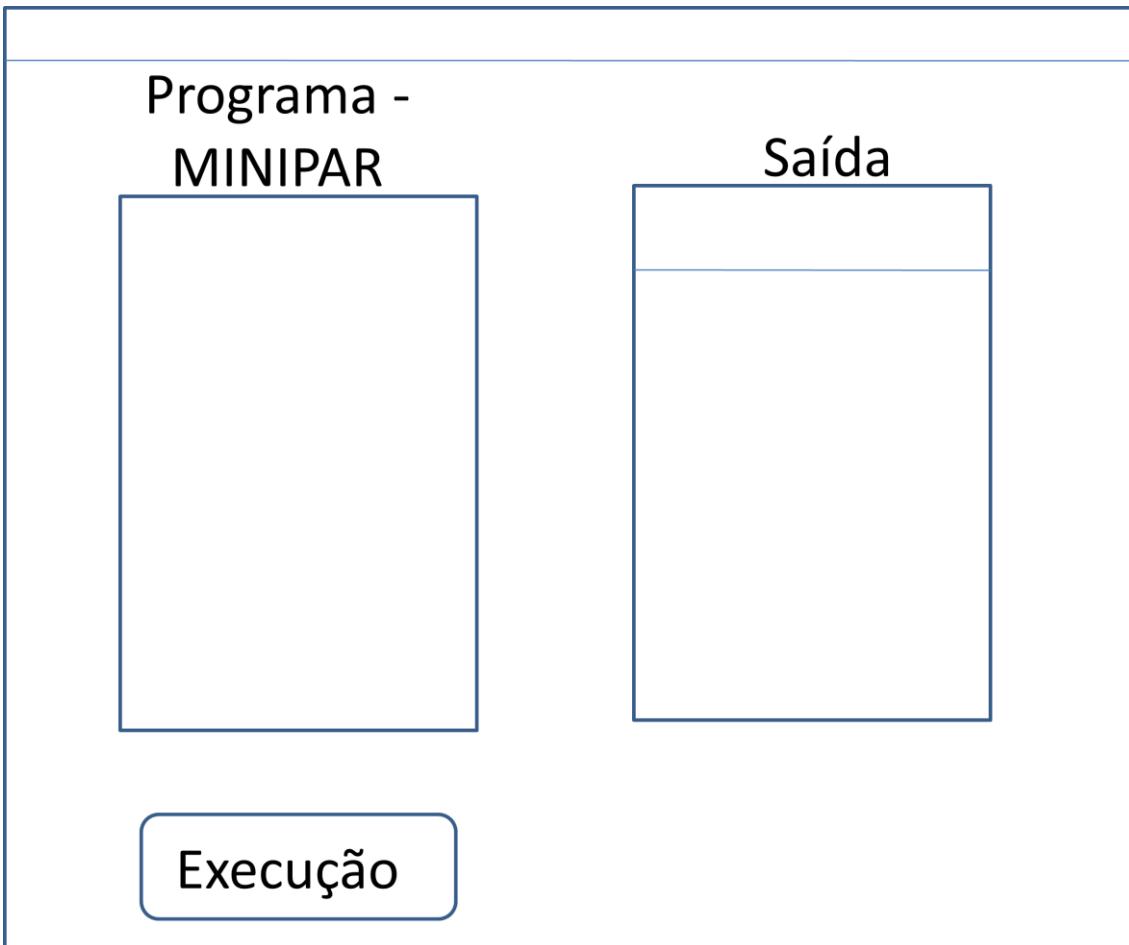
18. Colocar, no Github, o código fonte, os programas de testes e o relatório do Compilador da Linguagem Minipar 2025.1 utilizando Componentes de Software.

19 – Colocar o link do código fonte no Github da Linguagem Minipar 2025.1, utilizando Componentes, no relatório do Compilador da Linguagem MINIPAR 2025.1.

Apêndice 1

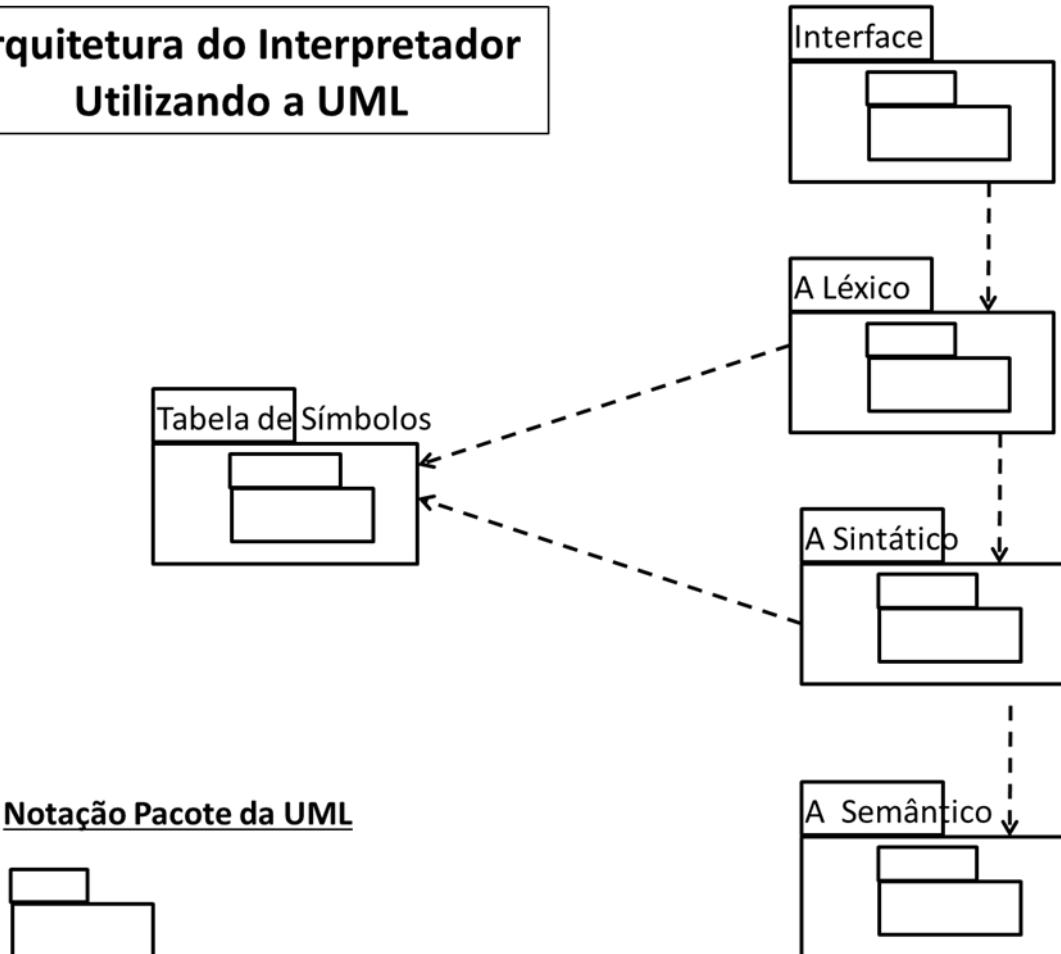
**Modelagem do Compilador para a Linguagem MINIPAR
versão 2025.1 utilizando Componentes de Software**

Tela do Compilador da Linguagem MINIPAR



MODELAGEM DO COMPILADOR DA LINGUAGEM MINIPAR versão 2025.1

Arquitetura do Interpretador Utilizando a UML

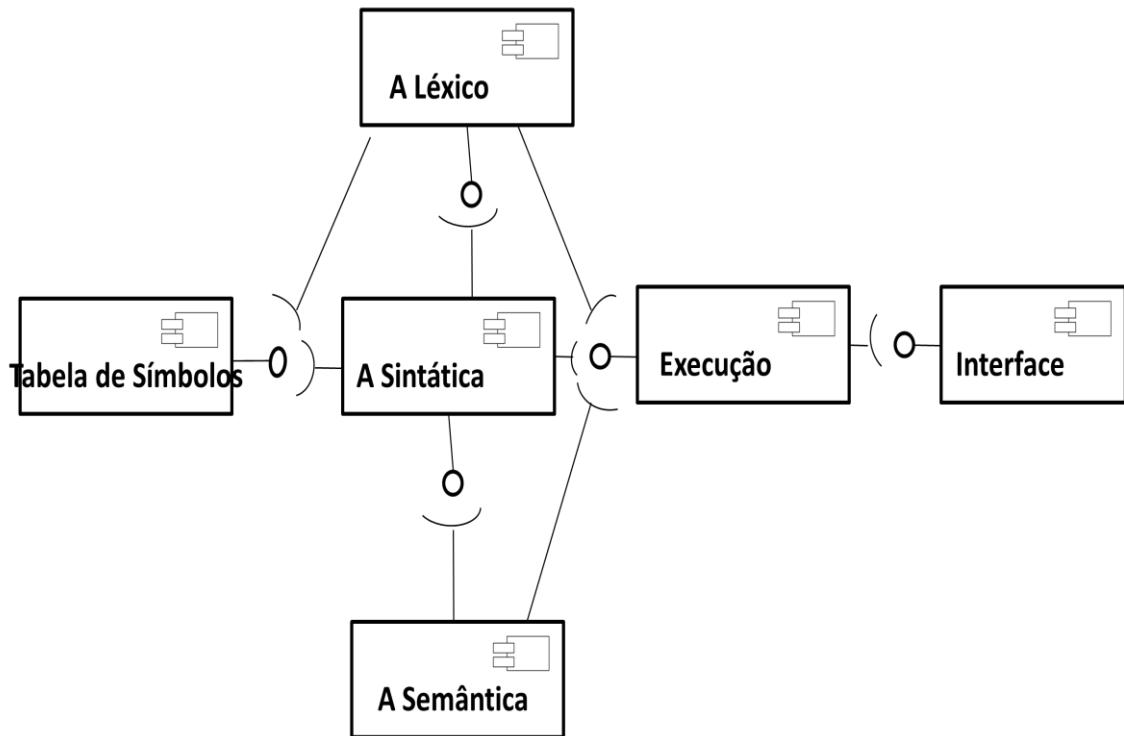


Observação:

Acrescentar os Pacotes para os Componentes de Software seguintes:

- Gerador de Código Intermediário de Três Endereços
- Gerador de Código Assembly para ARMv7

Diagrama de Componentes do Interpretador Utilizando a UML no Contexto da Linguagem MINIPAR



Observação:

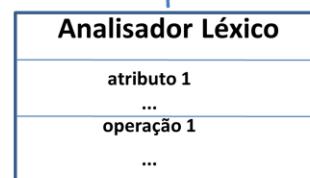
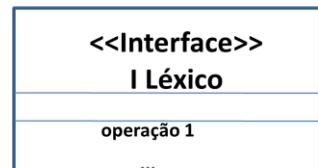
Acrescentar os Componentes de Software seguintes:

- Gerador de Código Intermediário de Três Endereços
- Gerador de Código Assembly para ARMv7

Cada Componente do Diagrama de Componentes do Interpretador deve ser Projetado via um Diagrama de Classes

Exemplo:

- Modelagem via Diagrama de Classes do Componente Analisador Léxico



IMPORTANTE:

Dessa forma para cada Componente de Software do Interpretador da Linguagem MINIPAR deve ser elaborado um diagrama de classes com a interface e a(s) classe(s) que implementa(m) essa interface

Apêndice 2

Exemplo de Modelagem de um Componente

Representar um relógio através de um componente de software usando a notação UML.

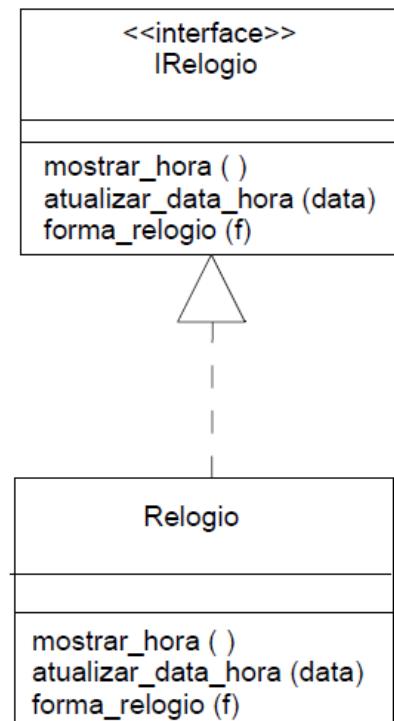


As operações associadas a interface fornecida IRelogio do componente Relogio são:

```
mostrar_hora()
atualizar_data_hora(data)
forma_relogio(f)
    f pode ser 1 - digital ou 0 - analógico
```

Modelagem do Componente Relogio (Diagrama de classes)

IRelogio – representa uma interface fornecida



Relogio – representa a classe que implementa (realiza) a interface IRelogio

A classe Relogio deve fornecer a implementação de cada operação (`mostrar_hora ()` – `atualizar_data_hora (data)` – `forma_relogio (f)`) da Interface IRelogio.

Exemplo de Implementação do Componente Relógio

Implementação em Java:

```
abstract class RelogioAbstrato {  
    // Métodos abstratos: devem ser implementados pelas subclasses  
    public abstract void atualizar_data_hora(String data);  
    public abstract void forma_relogio(String f);  
    public abstract void mostrar_hora();  
}  
  
class Relogio extends RelogioAbstrato {  
    private String dataHora;  
    private String formato;  
    @Override  
    public void mostrar_hora() {  
        System.out.println("Hora atual: " + dataHora + " (Formato: " + formato + ")");  
    }  
    @Override  
    public void atualizar_data_hora(String data) {  
        this.dataHora = data;  
        System.out.println("Data e hora atualizadas para: " + dataHora);  
    }  
    @Override  
    public void forma_relogio(String f) {  
        this.formato = f;  
        System.out.println("Formato do relógio definido para: " + formato);  
    }  
}
```

```
public class Aplicacao {  
    public static void main(String[] args) {  
        // Usando a classe abstrata como tipo de referência  
        RelogioAbstrato meuRelogio = new Relogio();  
  
        // Atualiza a data e hora  
        meuRelogio.atualizar_data_hora("14/08/2025 08:00");  
  
        // Define o formato do relógio  
        meuRelogio.forma_relogio("24h");  
  
        // Mostra a hora atual  
        meuRelogio.mostrar_hora();  
    }  
}
```

Execução (Saída):

Data e hora atualizadas para: 14/08/2025 08:00

Formato do relógio definido para: 24h

Hora atual: 14/08/2025 08:00 (Formato: 24h)

...Program finished with exit code 0

Press ENTER to exit console.