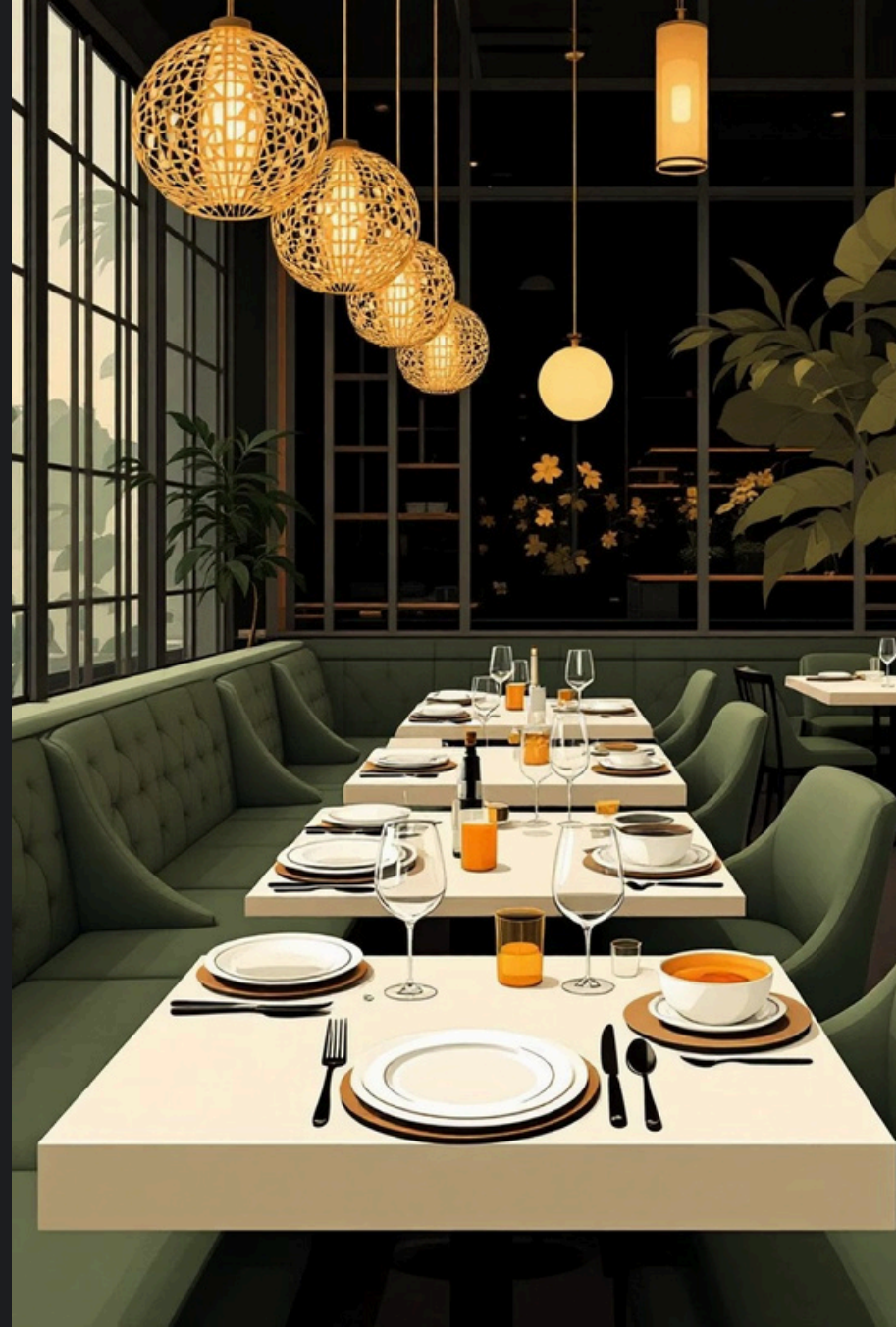
restLang Þ

Umalinguagem dedomínio específico para gerenciamento de restaurantesdesenvolvida com Flex e Bison

Autor: Gustavo Mendes | **Disciplina:** Lógica Computacional - APS



Motivação e Propósito

Por que restLang?

O**restLang** nasceudanecessidade de criar uma linguagem específica para simular operações de restaurantes no estilo **self-service e rodízio**.

Ao invés de usar linguagens de propósito geral, esta DSL oferece sintaxe natural e intuitiva para o domínio de restaurantes, facilitando cálculos de comandas, gestão de pedidos e regras de negócio.

Domínio Específico

Focado exclusivamente em operações de restaurantes

Sintaxe Clara

Comandos em português fáceis de entender

Implementação Rápida

Desenvolvido com Flex e Bison

Características Principais

Gestão de Clientes

Registreclientes comtipos de atendimento diferenciados: self-service com cobrança por peso ou rodízio com valor fixo.

Cálculo por Peso

Sistemaautomáticode cálculo baseado no peso do prato, com preço configurável por quilograma.

Itens Adicionais

Adicione bebidasesobremesas à comanda com controle de quantidade e preços individuais.

Condicionais

Implementeregras de negócio com estruturas SE/ENTAO para lógica condicional personalizada.

Repetição

Executeblocos de código múltiplas vezes usando loops REPETIR/VEZES de forma simples e intuitiva.

Fechamento de Conta

Gere comandasformatadasautomaticamente com todos os detalhes do consumo e valor total.

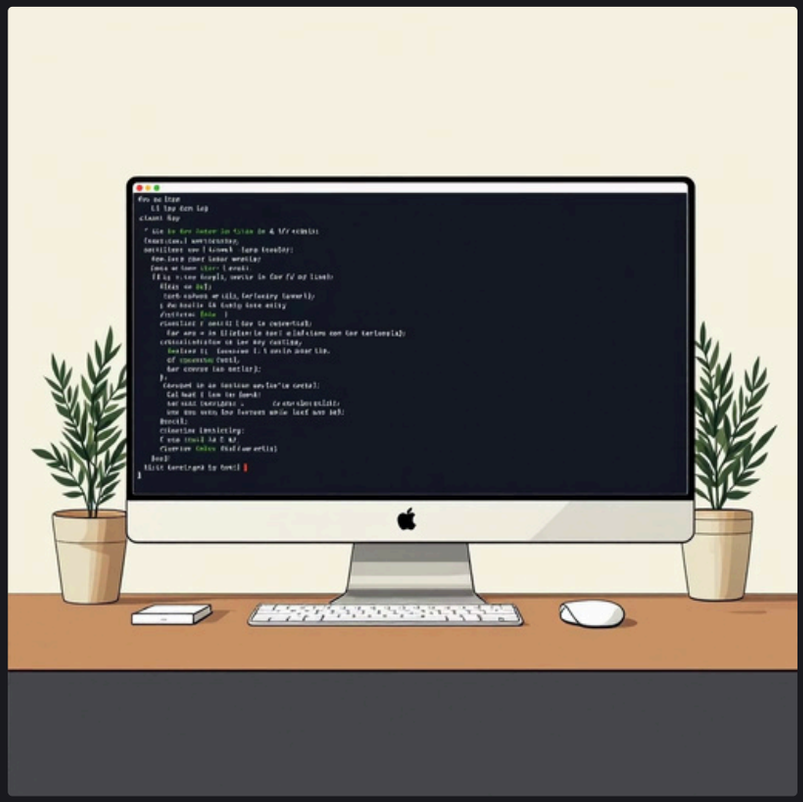
Requisitos Técnicos

Ferramentas Necessárias

ParacompilareexecutarorestLang,você precisará de um ambiente Linux ou WSL (Windows Subsystem for Linux) com as seguintes ferramentas instaladas:

Ferramenta	Versão Mínima	Função
GCC	7.0+	Compilador C
Flex	2.6+	Análise léxica
Bison	3.0+	Análise sintática
Make	4.0+	Automação build

Importante: Este projeto foi desenvolvido especificamente para ambientes Linux/WSL e não funcionará nativamente no Windows ou macOS sem adaptações.



Instalação Rápida

```
sudo apt update
sudo apt install gcc flex bison make
```

Como Usar

Navegue até o diretório

```
cd /mnt/c/caminho/para/restLang
```

Compile o projeto

```
make
```

Execute seu programa

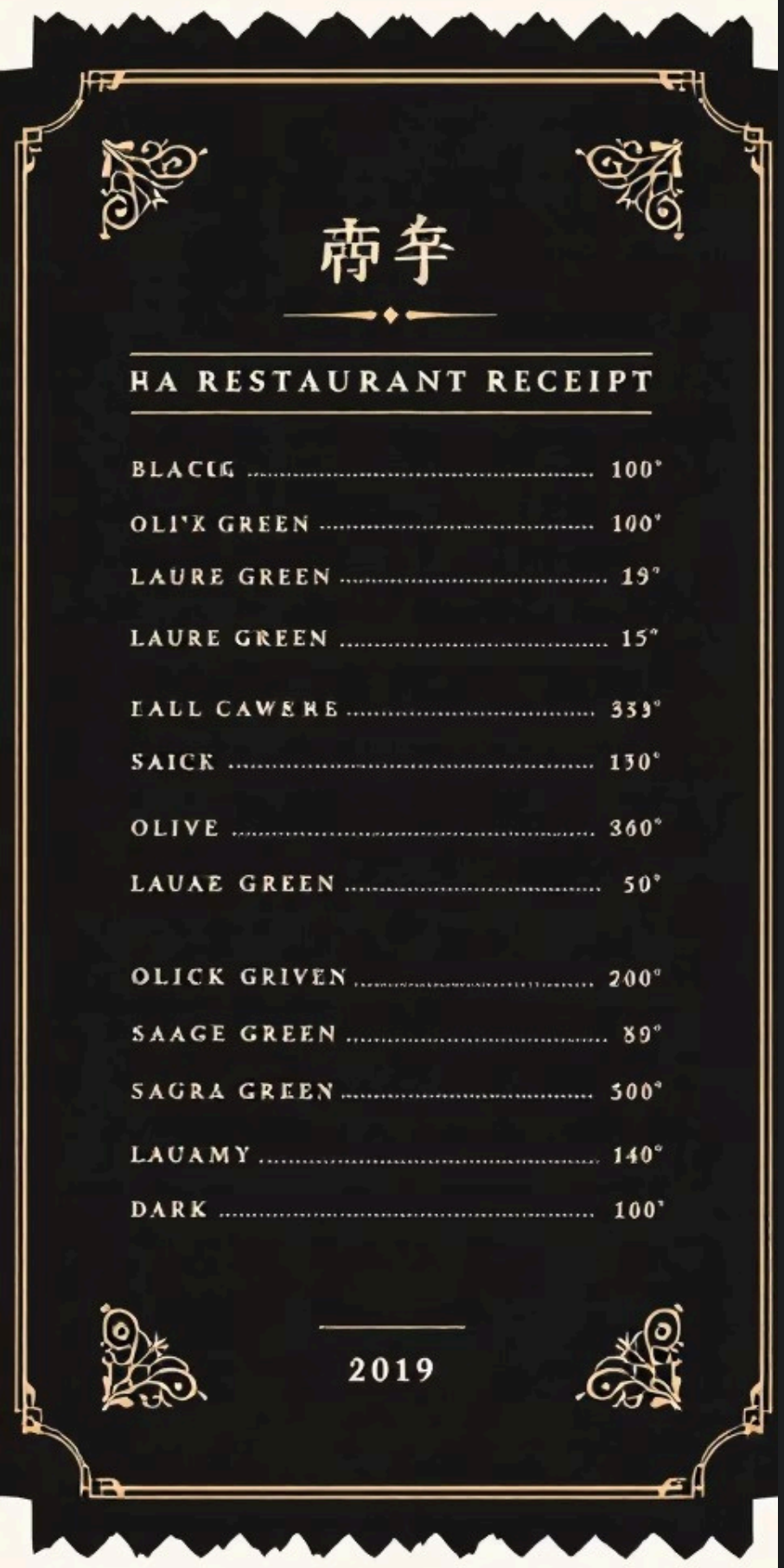
```
./rest_parser sample.rest && ./restvm
```

Ou rode todos os testes

```
make test-all
```

Workflow de Desenvolvimento

Oprocessoé simples: escrevaseucódigo em umarquivo `.rest`, compile com o parser que gera assembly, e execute na máquina virtual. O Makefile automatiza todo o processo de build.



Exemplo Prático

Código restLang

```
CLIENTE "Maria" {
tipo = SELF_SERVICE;
peso = 1.5;
 bebidas = 2;
 sobremesas = 1;
}

MOSTRAR "Bem-vinda!";

CALCULAR_PESO 1.0;

ADICIONAR_ITEM
"Refrigerante" 2;
SE cliente.peso > 1 ENTAO
{



  MOSTRAR "Prato
generoso!";
}

REPETIR 3 VEZES {
 MOSTRAR
"Processando...";
}
MOSTRAR cliente.total;
FECHAR_CONTA;
```

Saídado Programa

```
Bem-vinda! Prato generoso!
Processando...
Processando...
Processando...
============================
=========== CONTA DO CLIENTE
============================
===========                Maria
SELF_SERVICE ----------------------------
----------
Total: 45
============================
===========
```

Este exemplo demonstra todos os recursos principais: definição de cliente, cálculos automáticos, condicionais, loops e fechamento de conta formatado.

Sintaxe da Linguagem

1

Definição de Cliente

```
CLIENTE "Nome" {
tipo = SELF_SERVICE;
peso = 1.5;
bebidas = 2;
 sobremesas = 1;
}
```

2

Comandos de Exibição

```
MOSTRAR "texto";
MOSTRAR cliente.peso;
```

3

Operações de Cálculo

```
CALCULAR_PESO 1.5;
ADICIONAR_ITEM "Suco" 3;
```

4

Estruturas de Controle

```
SE cliente.peso > 2 ENTAO {
MOSTRAR "Muito!";
}
REPETIR 5 VEZES {
 MOSTRAR "Loop";
}
```

Operadores

- `=` Atribuição
- `==` Igualdade
- `>` Maior que
- `<` Menor que

Atributos do Sistema

- `sistema.preco_kilo`
- `sistema.preco_rodizio`
- `sistema.preco_bebida`

Atributos do Cliente

- `cliente.nome`
- `cliente.tipo`
- `cliente.peso`
- `cliente.total`

Arquitetura do Sistema

Código .rest

Arquivofonteescrito em restLang

Lexer (Flex)

Análise léxica e tokenização

Parser (Bison)

Análisesintáticaegeração de código

Assembly

Códigointermediário gerado

VM

Máquina virtual executa o código

Componentes Principais

lexer.l

Analisador léxico desenvolvido em Flex. Responsável por identificar tokens como palavras-chave, números, strings e operadores no código fonte.

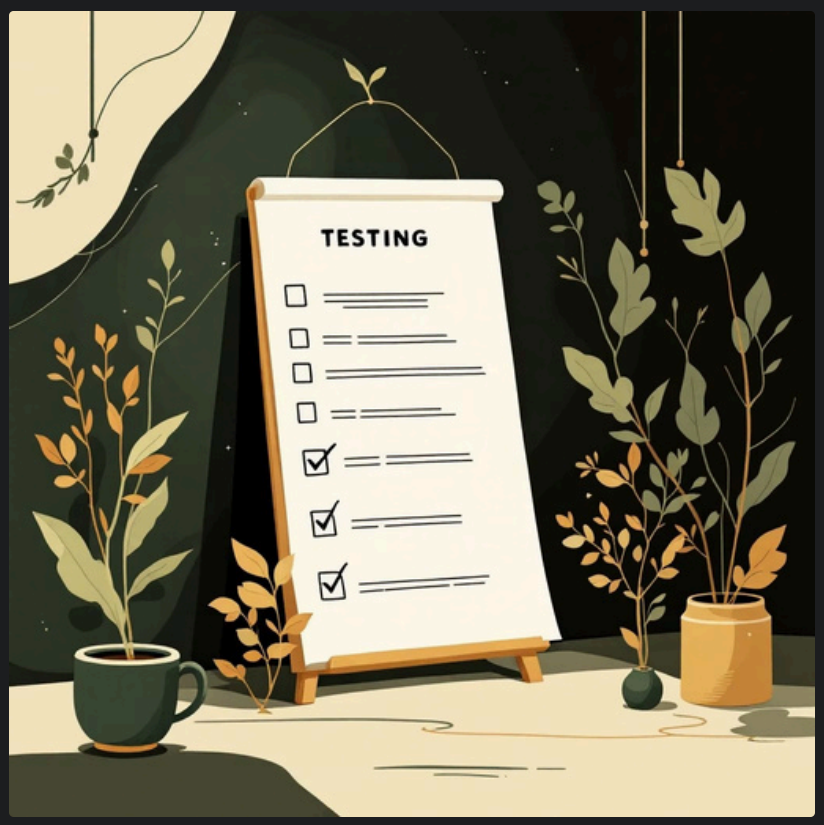
parser.y

Analisador sintático em Bison. Define a gramática da linguagem e gera código assembly intermediário para a máquina virtual.

restvm.c

Máquinavirtual implementada em C. Interpreta e executa o código assembly gerado pelo parser, produzindo a saída final.

Testes e Validação



Suite de Testes Disponíveis

Arquivo de Teste	Funcionalidade
test1_self_service.res	Cliente self-service completo
t test2_ rodizio.rest	Cliente tipo rodízio
test3_repetir.rest	Loop REPETIR X VEZES
test4_condicional.res	Estruturas SE/ENTAO
t test5_completo.rest	Todos os recursos integrados
test6_atribuicao.rest	Variáveis customizadas

Executar Testes

```
make test-all
```

ou individualmente:

```
./rest_parser tests/test1.rest && ./restvm
```

A suite de testes garante que todas as funcionalidades da linguagem estão operando corretamente, desde operações básicas até cenários complexos com múltiplos recursos combinados.

Gramática Formal EBNF

Estrutura do Programa

```
PROGRAMA = { COMANDO } ;

COMANDO =

  DEFINIR_CLIENTE
  | ADICIONAR
  | CALCULAR
  | FECHAR
  | MOSTRAR
  | BLOCO_SE
  | BLOCO_REPETIR
  | ATRIBUICAO ;

DEFINIR_CLIENTE =
  "CLIENTE" STRING "{"
    "tipo" "=" TIPO ";"
    "peso" "=" NUMBER ";"
    "bebidas" "=" NUMBER ";"
    "sobremesas" "=" NUMBER ";"
  "}" ;

TIPO = "SELF_SERVICE" | "RODIZIO" ;
```

Comandos e Expressões

```
ADICIONAR =
  "ADICIONAR_ITEM" STRING NUMBER ";" ;

CALCULAR = "CALCULAR_PESO" NUMBER ";" ;

FECHAR = "FECHAR_CONTA" ";" ;

MOSTRAR = "MOSTRAR" (STRING | ATRIBUTO) ";" ;

BLOCO_SE =

  "SE" CONDICAO "ENTAO"
  "{" {COMANDO} "}" ;

BLOCO_REPETIR =
  "REPETIR" NUMBER "VEZES"
  "{" {COMANDO} "}" ;

CONDICAO = ATRIBUTO OP VALOR ;

OP = ">" | "<" | "==" ;

ATRIBUTO = ("cliente"|"sistema") "." ID ;
```

A gramática completa está documentada em `docs/EBNF.md` e define formalmente todas as construções sintáticas válidas da linguagem restLang.

OBRIGADO!