Ciência da Computação
Instituto Federal do Ceará
Campus Maracanaú

Professor: Hericson Araújo
hericson.araujo@ifce.edu.br
Disciplina: Compiladores

Trabalho Final - Compiladores 2024.2

Desenvolvimento de um Tradutor de Linguagem Personalizada para C++ com Execução no ESP32

Objetivo Geral: Trabalhando em **duplas**, construa um tradutor de **linguagem personalizada** que converta programas escritos para **linguagem C++**, com foco em integração e execução no microcontrolador ESP32.

Objetivos Específicos:

1.) Compreender o funcionamento de um analisador léxico e sintático na construção de compiladores.

2.) Implementar regras léxicas para identificar os elementos da linguagem personalizada.

3.) Implementar regras sintáticas para validar e processar a estrutura do programa, traduzindo-o para C++.

4.) Incorporar ao código traduzido comandos específicos para o ESP32, como manipulação de GPIOs e periféricos.

5.) Compilar o código gerado em C++ e carregá-lo no ESP32 para executar as funcionalidades descritas.

6.) Realizar análise semântica.

O projeto possui as seguintes datas relevantes previstas:

- **20/01/2025 a 12/02/2025**: Finalização dos Projetos
- **16/02/2025**: Entrega do projeto final pelo *Classroom*
  - Código
  - Relatório
  - Apresentação
- **17/02/2025 e 19/02/2025**: Apresentação dos trabalhos
- **21/02/2025**: Prova Final

1 Características da Linguagem Personalizada

A linguagem personalizada incluirá comandos básicos para controle de dispositivos conectados ao ESP32, como LEDs, sensores e displays. Ela terá sintaxe simples e comandos de alto nível que serão traduzidos para C++.

1.1 Estrutura Básica do Programa

Um programa na linguagem personalizada é composto por dois blocos principais:

1.) Função `config`**:** Executada uma única vez no início do programa. Utilizada para inicializar variáveis, configurar pinos GPIO, PWM, Wi-Fi, ou qualquer outro elemento necessário antes da execução do loop principal.

2.) Bloco `repita ... fim`**:** O loop principal do programa. Contém o código que será executado continuamente enquanto o microcontrolador estiver ligado.

Abaixo, apresentamos a estrutura básica do programa na linguagem personalizada:

1.2 Exemplo de Código na Linguagem Personalizada

```
// Declaração de variáveis
var inteiro: ledPin, brilho;
var texto: ssid, senha;

// Função de setup
config
    ledPin = 2; // Configuração do pino onde o LED está conectado
    ssid = "MinhaRedeWiFi"; // Nome da rede Wi-Fi
    senha = "MinhaSenhaWiFi"; // Senha da rede Wi-Fi

    configurar ledPin como saida; // Configura o pino como saída
    configurarPWM ledPin com frequencia 5000 resolucao 8; // Configura PWM

    conectarWifi ssid senha; // Conecta à rede Wi-Fi
fim

// Loop principal
repita
    brilho = 128; // Define o valor do PWM (50% de intensidade)
    ajustarPWM ledPin com valor brilho; // Liga o LED com brilho definido
    esperar 1000; // Pausa por 1 segundo

    brilho = 0; // Define o valor do PWM como 0
```

```
    ajustarPWM ledPin com valor brilho; // Desliga o LED
    esperar 1000; // Pausa por 1 segundo
fim
```

1.3 Tipos de Dados e Variáveis

A linguagem permite declarar variáveis de diferentes tipos para armazenar informações e controlar dispositivos conectados ao ESP32.

Tipos de Variáveis

- **inteiro**: Representa valores numéricos inteiros.
  - Exemplo: `var inteiro: ledPin, brilho;`
- **booleano**: Representa estados lógicos (`true` ou `false`).
  - Exemplo: `var booleano: estadoBotao;`
- **texto**: Representa cadeias de caracteres (strings), usado para Wi-Fi ou mensagens.
  - Exemplo: `var texto: ssid, senha;`

Declaração de Variáveis

Variáveis são declaradas com o comando `var`, seguido pelo tipo, uma lista de identificadores, e finalizado por ponto e vírgula:

```
var inteiro: ledPin, brilho, botao;
var texto: ssid, senha;
```

Atribuição

Uma variável pode receber um valor com o operador =

```
ledPin = 2;
ssid = "MinhaRedeWiFi";
brilho = 128;
```

1.4 Controle de GPIO

Comandos para configurar e controlar pinos GPIO, permitindo leitura e escrita em dispositivos digitais.

Configuração de Pinos

Configura pinos GPIO como entrada (`entrada`) ou saída (`saida`):

```
configurar ledPin como saida;
configurar botao como entrada;
```

Controle de Estados Lógicos

Liga ou desliga pinos configurados como saída:

```
ligar ledPin;
desligar ledPin;
```

Leitura de Pinos

- **lerDigital**: Lê o estado lógico de um pino configurado como entrada:

  ```
  estadoBotao = lerDigital botao;
  ```

- **lerAnalogico**: Lê o valor analógico de um pino configurado para ADC:

  ```
  sensorValor = lerAnalogico sensor;
  ```

1.5 Controle de PWM

Permite gerar sinais PWM para controlar dispositivos como LEDs ou motores.

Configuração de PWM

Configura um pino GPIO para operar no modo PWM, especificando a frequência e a resolução:

```
configurarPWM ledPin com frequencia 5000 resolucao 8;
```

Ajuste de PWM

Ajusta o valor do PWM de um pino:

```
ajustarPWM ledPin com valor 128; // 50% do brilho (resolução de 8 bits)
```

1.6 Conexão Wi-Fi

Permite configurar e conectar o ESP32 a uma rede Wi-Fi, facilitando a comunicação via internet.

Configuração de Wi-Fi

Conecta o ESP32 a uma rede Wi-Fi usando o SSID e a senha:

```
ssid = "MinhaRedeWiFi";
senha = "MinhaSenhaWiFi";
conectarWifi ssid senha;
```

Envio de Dados via HTTP

Envia dados para um servidor HTTP com um método simples:

```
enviarHttp "http://example.com" "dados=123";
```

1.7 Comunicação Serial

Permite configurar a comunicação serial e enviar ou receber dados via porta serial.

Configuração da Serial

Configura a velocidade de comunicação serial (baud rate):

```
configurarSerial 115200;
```

Envio e Recepção de Dados

- **escreverSerial**: Envia mensagens pela serial:

  ```
  escreverSerial "Botão pressionado!";
  ```

- **lerSerial**: Lê mensagens recebidas pela serial:

  ```
  mensagemRecebida = lerSerial;
  ```

1.8 Controle de Fluxo

Permite executar trechos de código condicionalmente ou em loops.

6.1. Condicional (`se` e `senao`)

Executa comandos se uma condição for verdadeira, com suporte para `senao`:

```
se estadoBotao == 1 entao
    ligar ledPin;
senao
    desligar ledPin;
fim
```

Laço (`enquanto`)

Repete comandos continuamente:

```
enquanto
    ajustarPWM ledPin com valor brilho;
    esperar 1000;
fim
```

1.9 Delays

Aguarda um tempo definido antes de continuar a execução do programa.

```
esperar 1000; // Aguarda 1 segundo
```

1.10 Expressões e Operadores

Permite usar operações lógicas e aritméticas em condições e atribuições.

Operadores Relacionais

- `==`: Igual a.
- `!=`: Diferente de.
- `<`: Menor que.
- `>`: Maior que.
- `<=`: Menor ou igual.
- `>=`: Maior ou igual.

Operadores Aritméticos

- `+`: Soma.
- `-`: Subtração.
- `*`: Multiplicação.
- `/`: Divisão.

Exemplo de Uso de Expressões

```
se brilho > 128 entao
    ligar ledPin;
fim
```

1.11 Exemplo de Código na Linguagem Personalizada

```
// Declaração de variáveis
var inteiro: ledPin, brilho;
var texto: ssid, senha;

// Função de setup
config
    // Atribuição de valores às variáveis
```

```
    ledPin = 2; // Pino GPIO onde o LED está conectado
    ssid = "MinhaRedeWiFi"; // Nome da rede Wi-Fi
    senha = "MinhaSenhaWiFi"; // Senha da rede Wi-Fi

    // Configuração de pinos
    configurar ledPin como saida;
    configurarPWM ledPin com frequencia 5000 resolucao 8;

    // Configuração de Wi-Fi
    conectarWifi ssid senha;
fim

// Loop principal
repita
    brilho = 128; // Valor do PWM (0-255 para resolução de 8 bits)
    ajustarPWM ledPin com valor brilho; // Liga o LED com 50% de brilho

    esperar 1000; // Aguarda 1 segundo

    brilho = 0; // Desliga o LED
    ajustarPWM ledPin com valor brilho; // Ajusta o PWM para 0 (LED apagado)

    esperar 1000; // Aguarda 1 segundo
fim
```

1.12 Código da subseção 1.9 Traduzido para C++

```
#include <Arduino.h>
#include <WiFi.h>

// Declaração de variáveis
int ledPin;               // Pino GPIO onde o LED está conectado
int brilho;               // Valor do PWM (0-255)
String ssid;              // Nome da rede Wi-Fi
String senha;             // Senha da rede Wi-Fi

// Configuração do PWM
const int canalPWM = 0;    // Canal do PWM para o pino
const int frequencia = 5000; // Frequência do PWM em Hz
const int resolucao = 8;    // Resolução do PWM em bits (0-255)
```

```
void setup() {
    // Atribuição de valores às variáveis
    ledPin = 2;
    ssid = "MinhaRedeWiFi";
    senha = "MinhaSenhaWiFi";

    // Configuração do pino como saída
    pinMode(ledPin, OUTPUT);

    // Configuração do PWM
    ledcSetup(canalPWM, frequencia, resolucao);
    ledcAttachPin(ledPin, canalPWM);

    // Conexão ao Wi-Fi
    WiFi.begin(ssid.c_str(), senha.c_str());
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println("Conectando ao WiFi...");
    }
    Serial.println("Conectado ao WiFi!");
}

void loop() {
    // Liga o LED com 50% de brilho (PWM = 128)
    brilho = 128;
    ledcWrite(canalPWM, brilho);
    delay(1000); // Aguarda 1 segundo

    // Desliga o LED (PWM = 0)
    brilho = 0;
    ledcWrite(canalPWM, brilho);
    delay(1000); // Aguarda 1 segundo
}
```

O Trabalho Final consiste em **Relatório** e **Execução** das questões a seguir:

(1) Parte 1: Análise Léxica e Sintática

(a) **Tradução para C++:** A linguagem personalizada deverá ser traduzida para código C++ compatível com o ESP32, utilizando bibliotecas padrão como a WiFi.h e Arduino.h.

(b) **Implementação do Tradutor:** O tradutor será implementado usando **Flex e Bison**. Ele será dividido em **analisador léxico**, **analisador sintático** e **gerador de código**.

(c) Caso haja algum erro léxico,sintático ou semântico, deverá ser informado.

(d) Mostre a **árvore sintática** criada para cada análise

(e) Crie um arquivo **.sh** que automatize o processo.

(f) Gere um arquivo em **linguagem C++** a partir do código programado na **linguagem personalizada**.

(g) Crie três programas de exemplo em **linguagem personalizada**, converta para **linguagem C++** e teste seu código no **ESP32**. Você também poderá usar um simulador online: wokwi.

(2) Parte 2: Análise Semântica

- **Declaração e Uso de Variáveis:**

  1.) Todas as variáveis usadas no programa devem ser previamente declaradas.

  2.) Não é permitido declarar uma variável com o mesmo nome duas vezes no mesmo escopo

- **Exemplo:**

```
// Função de setup
config
    configurar ledPin como saida; // Configuração do p
fim

// Loop principal
```

```
repita
    ligar ledPin; // Liga o LED conectado ao pino
    esperar 1000; // Pausa por 1 segundo
    desligar ledPin; // Desliga o LED
    esperar 1000; // Pausa por 1 segundo
fim
```

Erro semântico: Variável 'ledPin' não foi declarada antes do uso.

- **Tipos de Variáveis:**

  1.) Operações realizadas devem ser compatíveis com os tipos das variáveis (por exemplo, não se pode usar operações aritméticas com variáveis do tipo texto).

  2.) Os valores atribuídos às variáveis devem ser do tipo correto.

- **Exemplo:**

```
// Declaração de variáveis
var texto: brilho;

// Configuração de PWM
config
    brilho = 128;
    configurarPWM ledPin com frequencia 5000 resolucao 8;
fim
```

Erro Semântico: Tentativa de usar 'texto' para armazenar um valor numérico

- **Compatibilidade de Comandos:**

  1.) Comandos como ligar, desligar, ajustarPWM só podem ser usados com pinos configurados como saida ou PWM.

  2.) Comandos como lerDigital ou lerAnalogico só podem ser usados com pinos configurados como entrada.

- **Exemplo:**

```
var inteiro: ledPin, sensor;
```

```
config
    configurar ledPin como saida;
fim
repita
    ajustarPWM ledPin com valor 128;
    estado = lerDigital sensor;
fim
```

Erro Semântico: ledPin não está configurado como PWM

Erro Semântico: sensor não foi configurado como entrada