



APS-LINGUAGEM: GardenGo



por Yan Romano



Resumo do Projeto

Este projeto apresenta o desenvolvimento completo de "GardenGo", uma linguagem de programação de domínio específico (DSL) criada para a automação de jardins inteligentes. O trabalho abrange desde a especificação formal da linguagem com EBNF, passando pela implementação de um compilador utilizando as ferramentas Flex e Bison, até a geração de código intermediário (IR) para a Máquina Virtual LLVM, com execução Just-in-Time (JIT). O resultado é um sistema funcional capaz de traduzir comandos de alto nível em código executável, que interage com um runtime simulado para executar tarefas como plantar, regar e tomar decisões baseadas em sensores.

Motivação e Objetivos

Motivação (O "Porquê"):

A motivação para o projeto surge de um desafio prático e pessoal: a dificuldade em manter a consistência nos cuidados com plantas, uma tarefa que se beneficia enormemente da automação. A proposta foi criar uma ferramenta que permitisse a qualquer pessoa, mesmo sem conhecimento aprofundado em programação de baixo nível ou eletrônica, escrever um roteiro lógico e legível para automatizar um jardim.

Objetivos do Projeto (O "O Quê"):

Os objetivos seguiram estritamente o escopo definido para a avaliação:

- Criar uma Linguagem de Programação:** A linguagem deveria conter as estruturas básicas de variáveis (implícitas), condicionais (IF/ELSE) e laços de repetição (LOOP).
- Especificação Formal:** Estruturar a gramática da linguagem segundo o padrão EBNF.
- Implementação do Compilador:** Utilizar Flex e Bison para análise léxica e sintática, e integrar com uma VM (LLVM) para a geração de código e execução.
- Demonstração:** Criar um programa de teste completo que demonstrasse todas as características da linguagem.

Características da Linguagem

A GardenGo foi projetada com os seguintes princípios em mente:

Comandos Intuitivos:

A sintaxe utiliza verbos em inglês que representam ações diretas, tornando o código legível e fácil de entender.

Exemplo: PLANT Tomate AT (1,1)

Gerenciamento Espacial:

A linguagem permite a definição de ZONES de cultivo e a execução de comandos em COORDenadas específicas, refletindo o layout físico de um jardim.

Exemplo: ZONE HortaSul = (6,0 - 10,5)

Lógica Condisional:

Decisões podem ser tomadas com base em dados de sensores (até agora simulados), como clima e umidade do solo, permitindo uma automação "inteligente".

Exemplo: IF WEATHER.TEMPERATURE > 20 { ... }

Controle de Fluxo:

O suporte a laços de repetição (LOOP N TIMES) permite a criação de ciclos de manutenção (diários, semanais), estabelecendo a linguagem como computacionalmente completa para seu domínio.

Exemplo: LOOP 3 TIMES { WAIT 1h }

Arquitetura do Compilador GardenGo

Fluxograma: Código .garden → [Analizador Léxico (Flex)] → Fluxo de Tokens → [Analizador Sintático (Bison)] → Árvore Sintática Abstrata (AST) → [Gerador de Código (codegen.c)] → LLVM IR → [Motor JIT da LLVM] → Execução

Descrição das Etapas:



Análise Léxica (Flex):



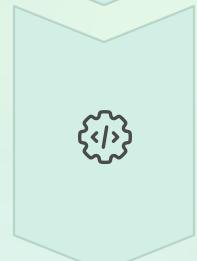
O arquivo scanner.l define regras para converter o código-fonte em um fluxo de tokens (palavras-chave como PLANT, números, identificadores).



Análise Sintática (Bison):



O parser.y implementa a gramática EBNF. Ele consome os tokens e constrói uma Árvore Sintática Abstrata (AST), que é a representação do programa em memória.



Geração de Código (codegen.c):



Este componente C percorre a AST recursivamente e traduz cada nó (comando, condição, laço) em instruções de baixo nível para a LLVM.



Backend LLVM (JIT):



O código intermediário (LLVM IR) gerado é passado para o motor Just-In-Time da LLVM, que o compila para código de máquina nativo e o executa imediatamente, interagindo com as funções de runtime em C.

Geração de Código: Um Estudo de Caso

Para ilustrar o processo de compilação, analisamos como o comando LOOP de alto nível é traduzido em uma estrutura de controle de baixo nível na LLVM.

Código GardenGo (Entrada):

```
LOOP 3 TIMES {  
    WAIT 1s  
}
```

LLVM IR (Saída Gerada):

Snippet de código entry:

```
%loop_counter = alloca i32 ; 1. Aloca variável para o contador  
store i32 0, ptr %loop_counter ; 2. Inicializa contador com 0  
br label %loop_cond ; 3. Pula para a condição loop_cond:  
%load_count = load i32, ptr %loop_counter ; 4. Carrega o valor  
%loop_cmp = icmp slt i32 %load_count, 3 ; 5. Compara: contador < 3  
br i1 %loop_cmp, label %loop_body, label %loop_exit ; 6. Desvio cond.  
loop_body:  
call void @garden_wait(i32 1)  
br label %loop_inc  
loop_inc:  
%load_old_count = load i32, ptr %loop_counter  
%add_count = add i32 %load_old_count, 1 ; 7. Incrementa contador  
store i32 %add_count, ptr %loop_counter  
br label %loop_cond ; 8. Volta para a condição loop_exit:  
; ... continua o programa
```

Demonstração Completa e Execução

Código-Fonte de Teste:

```
ZONE HortaNorte = (0,0 - 5,5)
IF WEATHER.TEMPERATURE > 20 {
    PLANT Tomate AT (1,1) IN HortaNorte
} ELSE {
    PLANT Alface AT (7,1)
}
LOOP 2 TIMES {
    WAIT 5s
    FERTILIZE NPK_10_10_10 AT (1,1) FOR 10s
}
HARVEST Tomate FROM HortaNorte
```

Saída da Execução (JIT):

```
--- Executando o código com JIT ---
[Runtime] Zona 'HortaNorte' declarada de (0,0) a (5,5).
[Runtime] Lendo sensor de temperatura... (retornando
30 para teste)
[Runtime] Plantando 'Tomate' na coordenada (1,1)
dentro da zona 'HortaNorte'.
[Runtime] Esperando por 5 segundos...
[Runtime] Fertilizante 'NPK_10_10_10' aplicado em (1,1)
por 10s.
[Runtime] Esperando por 5 segundos...
[Runtime] Fertilizante 'NPK_10_10_10' aplicado em (1,1)
por 10s.
[Runtime] Colhendo 'Tomate' da zona 'HortaNorte'.
--- Execução Concluída ---
```

Análise:

A saída da execução demonstra o funcionamento de todas as funcionalidades. O compilador avaliou a condição IF e executou o bloco then. Em seguida, entrou no laço LOOP por duas iterações, validando a arquitetura completa do compilador.

Conclusão

O projeto serviu para utilizar os conceitos aprendidos em aula e expandi-los.

Trabalhos Futuros:

- **Integração com Hardware:** Adaptar o garden_runtime.c para interagir com GPIOs de um Raspberry Pi ou Arduino, controlando um jardim real.
- **Compilação Ahead-of-Time (AOT):** Gerar um arquivo executável autônomo (.exe ou ELF) em vez de usar o JIT.
- **Expansão da Linguagem:** Adicionar suporte a variáveis e funções definidas pelo usuário para permitir lógicas mais complexas.



Repositório do Projeto

O código-fonte completo, incluindo o compilador (Flex, Bison, C), a infraestrutura da LLVM, o Makefile, os exemplos de teste e esta apresentação, está disponível no seguinte repositório Git:

<https://github.com/yanorck/GardenGo>