siC: Uma linguagem baseada em C incluindo pilha e fila como tipos primitivos

Gabriella de Oliveira Esteves, 110118995

[1]Departamento de Ciência da Computação - Universidade de Brasília

1. Objetivo

Este trabalho visa projetar e construir uma nova linguagem chamada de siC - Structures in C, baseada na linguagem C. O siC acrescenta três estruturas de dados, pilha e fila, como tipos de dados primitivos e, para manipulá-las, adiciona certas operações próprias para tal.

2. Introdução

Um compilador é um programa que recebe como entraga um código fonte e o traduz para um programa equilavente em outra linguagem [1]. Ele pode ser dividido em sete fases, ilustrado na Figura 1.

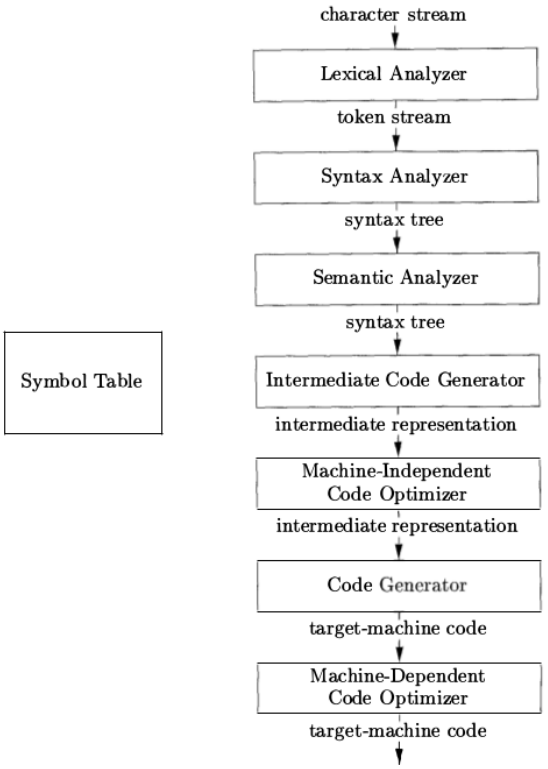

Figura 1. Fases de um compilador

1 **Analisador Léxico:** Lê o código fonte e atribui significado à cada sequência de caracteres, agora chamados lexemas. Cada lexema é mapeado para um token, que por sua vez é um par de nome (símbolo abstrato) e atributo (ponteiro para tabela de símbolos);

2 **Analisador Sintático:** Constrói uma representação gramátical dos tokens em forma de árvore;

3 **Analisador Semântico:** Utiliza a árvore sintática juntamente com a tabela de símbolos para verificar se a consistência semântica é mantida de acordo com a definição da linguagem.

4 **Gerador de Código Intermediário:** Converte árvore sintática anotada em código intermediário, com linguagem parecida com assembly e que possui apenas três operadores por linha de código. Nesse sentido, quebra-se estruturas complexas em estruturas mais simples, nesta fase.

5, 6 **Otimizador de Código Independente/Dependente de Máquina:** Procura aprimorar o código intermediário com o objetivo de melhorar o código-alvo de alguma forma: o deixando mais rápido, mais curto, consumindo menos energia, etc.

7 **Gerador de Código:** Converte o código intermediário no código-alvo, buscando atribuir os registradores às variáveis da maneira ótima.

O foco do projeto será nas fase 1, 2, 3 e 4, porém a princípio serão apresentadas apenas a descrição da linguagem siC, bem como uma breve descrição de sua semântica.

Dois grandes motivos sustentam a escolha do tema deste projeto. Primeiro, uma vez que as estruturas de dados pilha e fila fazem parte dos tipos primitivos de uma linguagem, haverá menos manipulação de ponteiros na mesma, portanto erros envolvendo-os são menos prováveis de ocorrer. Segundo, a linguagem siC é mais alto-nível que C devido à abstração das estruturas de dados báscias, e, de maneira geral, pode ser mais *user-friendly*. Nesse sentido, o usuário (da linguagem) leigo deverá entender como cada estrutura funciona, bem como suas vantagens/desvantagens e usabilidade; porém a implementação de cada uma estará a cargo da própria siC.

3. Gramática

A seguir será apresentada a gramática da linguagem siC, que é unicamente baseada em C [2]. Alguns comentários são feitos ao longo da gramática para facilitar o entendimento das variáveis e nomenclatura utilizada. As palavras reservadas da linguagen são representadas aqui como *tokens*. As variáveis e constantes são representadas como *identifiers* e a única diferença entre este e *identifier_new* é que o segundo tem acesso ao topo da pilha ou início da fila caso estes sejam os tipos do *identifier*. Segue abaixo a gramática proposta.

```
% token: WHILE, IF, ELSE
% token: TOP, FIRST

function
    : type identifier_char '(' argument ')' statement
    ;

identifier
    : identifier_char
    | identifier_digit
    ;
```

```
identifier_char
    : identifier_char
    | 'a' | 'b' | 'c' | ... | 'z'
    ;

identifier_digit
    : identifier_digit
    | '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' |'8' | '9'
    ;

identifier_new
    : identifier
    | identifier '.' TOP
    | identifier '.' FIRST
    ;

type
    : simple_type
    | stack_type
    | queue_type
    ;

simple_type
    : void
    | bool
    | int
    | char
    ;
```

Existem dois novos tipos de dados, *stack* e *queue*, que serão compostas por tipos simples de dados apenas (ou seja, não será possível criar uma variável do tipo pilha onde seus elementos também são pilhas). Caso a variável identificada pelo token IDENTIFIER seja do tipo pilha, ela poderá obter o topo através do comando "identifier.TOP". Caso seja fila, poderá obter o primeiro elemento com "identifier.FIRST".

```
stack_type
    : stack '<' simple_type '>'
    ;

queue_type
    : queue '<' simple_type '>'
    ;
```

```
argument
    : type identifier
    | type identifier ',' type identifier
    ;

statement
    : basic_expressions
    ;
```

A seguir serão descritas quatro estruturas básicas da linguagem siC: comando com repetição, condicional, expressões matemáticas e expressões com pilhas e filas. A última contempla as operações de adicionar elemento no topo da pilha ou no fim da fila, "+", e remover do topo ou do início da fila, -", onde o valor do elemento retirado é armazenado no último operando da expressão.

```
basic_expressions
    : while_expression
    | if_expression
    | math_expression
    | identifier_new_expression
    ;

while_expression
    : WHILE '(' compare_expression ')' statement
    ;

if_expression
    : IF '(' compare_expression ')' statement
    | IF '(' compare_expression ')' statement ELSE statement
    ;

compare_expression
    : compared_identifier compare_assignment compared_identifier
    ;

compared_identifier
    : identifier_new
    | '(' compare_expression ')'
    ;

compare_assignment
    : '=='
    | '!='
    | '<='
    | '>='
    ;
```

```
math_expression
    : identifier '=' factor
    ;

factor
    : identifier_new
    | '(' number_expression ')'
    ;

term
    : term '*' factor
    | term '/' factor
    | factor
    ;

number_expression
    : number_expression '+' term
    | number_expression '-' term
    | term
    ;


identifier_new_expression
    : identifier '=' identifier '+' identifier_new
    | identifier '=' identifier '-' identifier_new
    ;
```

Referências

[1] A. V. Abo, M. S. Lam, R. Sethi, J. D. Ullman, *Compilers - Principles, Techniques and Tools* 2nd ed. 1986

[2] ANSI C Yacc grammar, http://www.quut.com/c/ANSI-C-grammar-y.html, 18 12 2012.