MOE: uma linguagem alto-nível para programas ACL

> Este programa se trata da implementação em estágio Alpha da linguagem de programação apelidada como Moe.

Propósito

O objetivo dessa linguagem é fornecer uma interface entre o programador e programas ACL que mais se aproxime de uma linguagem natural do que de uma linguagem de máquina.

Com seu compilador, é possível escrever programas ACL com uma sintaxe semelhante a de linguagens como C, Python e Fortran (suas principais inspirações).

Sintaxe

A sintaxe se assemelha muito à linguagem C. Dentre as semelhanças, pode-se listar:

- os símbolos de comparação (>=, <=, <, >, !=, ==);
- símbolos dos operadores aritméticos (+, -, *, /);
- símbolos dos operadores lógicos (&&, ||, !);
- o uso de ponto e vírgula para delimitar uma declaração;
- termos dados aos tipos de variáveis (int e string);
- estruturas de controle de fluxo (if, else, for).

Além disso, todo programa Moe válido precisa ser inicializado com a palavra reservada `program` seguida pelo seu nome e seu corpo.

```
program MeuPrograma
{
    ...
}
```

O nome do programa não pode ser mais utilizado durante o mesmo, uma vez que ele vira um símbolo registrado.

Não há funções ou objetos em Moe, o que significa que todo programa é um script que roda uma vez só até o final. Porém, ele possui alguns comandos pré-estabelecidos que podem aceitar a inserção de argumentos, apresentando uma sintaxe semelhante à de funções em C.

Tipos de dado

A linguagem atualmente comporta cinco tipos de dados: **números inteiros**, **strings**, **posições**, **parâmetros** e **booleanos**.

Números Inteiros

Conforme implementação da linguagem ACL original, em Moe é possível utilizar números inteiros como variáveis ou como constantes. Para criar uma variável de número inteiro, basta declará-la com a palavra-chave `int`, dar um nome a ela e, opcionalmente, inicializá-la com um valor.

```
int y;
int x = 0;
```

(Obs: todas as declarações devem ser feitas dentro do corpo do programa.)

Os números inteiros também podem ser usados em expressões comuns no mundo da programação, como operações aritméticas, comparações e até expressões lógicas.

```
int x = 1 + 1;
x = 2 * x;

int y = 6 / 3;

int z = 1 || 0;
z = 1 && z;

int w = 6 > 4;
w = 9 < 3;
w = x == 8;
```

Booleanos

Os valores booleanos são máscaras para os valores 0 e 1, os quais podem ser utilizados para validações de lógicas condicionais, por exemplo.

```
int x = true && true;

if (x)
{
    ...
}
```

Sua lógica ainda está sendo estruturada para que ele deixe de ser apenas uma máscara, mas eles já podem ser utilizados para que o código fique um pouco mais legível.

Strings

Moe também possui suporte para strings, mas não como variáveis armazenadas. Uma vez que não é possível declarar variáveis em ACL, elas aparecem apenas como literais.

```
print("podem ser utilizadas como texto printável");

"como também podem ser usadas como comentários";
"mas não se esqueça do ; no final! ->";
```

Posições

Outra estrutura que veio do ACL são as posições. No momento, elas aparecem apenas como variáveis que podem ser declaradas e cujo valor pode ser atribuído. Algumas funções, como HERE e SHIFT haverão de ser adicionadas posteriormente.

Para declarar uma posição, é necessário especificar o grupo ao qual ela pertence entre colchetes. Esse grupo pode ser A, B ou C.

```
position[A] p1;
position[B] p2;
position[C] p3;
```

Parâmetros

Parâmetros também podem ser manipulados em Moe. Entretanto, ainda é necessário que a diretiva PRIVILEDGE esteja ativada, e esta ainda não foi implementada.

Para mudar parâmetros, basta utilizar a palavra-chave `parameter`, seguida do número de identificação do parâmetro, um sinal de igual e o valor a ser atribuído.

```
parameter 73 = 9350;
```

Escopo de variáveis

Variáveis inteiras também podem ser declaradas como globais ou não. Para declarar um inteiro como `global`, basta inserir a palavra reservada antes da declaração da variável.

```
global int x;
```

Palavras-chave

Moe possui 18 palavras-chave até o momento, sendo elas:

- print: comando print;
- move: comando move;
- delay: comando delay;
- jaw: comando jaw;
- open: comando open;
- close: comando close;
- await: auxiliar do comando moved;
- int: declarador;
- position: declarador;
- parameter: declarador;
- program: declarador;
- global: declarador;
- true: valor booleano;
- false: valor booleano;
- for: estrutura de controle de fluxo;
- between: auxiliar do comando for;
- if: estrutura de controle de fluxo;
- else: estrutura de controle de fluxo.

Comandos e Estruturas

Moe possui alguns dos principais comandos da linguagem ACL conforme pode ser observado na seção de palavras-chave, além de contar também com algumas estruturas de controle de fluxo.

Comandos

Delay: retarda o programa em um número determinado de centésimos de segundos.

Recebe um valor inteiro que equivale ao tempo de retardo.

Print: exibe uma string na tela.

Recebe um literal de uma string.

```
print("teste123");
```

Move/Moved: move o robô até uma posição especificada. `MOVE` não obedece ao fluxo do restante do programa, enquanto que `MOVED` sim.

Para representar o comando `MOVED`, basta utilizar a palavra-chave auxiliar `await` antes do comando `move`.

Pode receber um ou dois valores: a posição final e a duração do movimento.

```
move(p1);

await move(p2, 300); "vai até p2 em 3 segundos, seguindo o fluxo do programa";
```

```
delay(100); "para por 1 segundo";

int x = 200;

delay(x); "para por 2 segundos";
```

Jaw: abre a garra em um valor determinado.

Recebe o tamanho da abertura e pode receber o tempo em que deve abrir.

```
jaw(40); "abre a garra em 40%";

jaw(0, 300); "fecha a garra em 3 segundos";
```

Open: abre a garra.

Pode receber a duração da abertura.

```
open();

open(400); "abre a garra em 4 segundos";
```

Close: fecha a garra.

Pode receber a duração do fechamento.

```
close();

close(500); "fecha a garra ao longo de 5 segundos";
```

Estruturas de controle

If/Else: estrutura condicional. Executa um bloco de código dependendo do resultado de uma expressão lógica.

```
if (1 > 2)
{
    print("um é maior que dois");
}
else if (2 + 2 == 5)
{
    print("dois mais dois é igual a cinco");
}
else
{
    print("vai cair aqui porque nenhuma das estruturas acima é válida.");
}
```

For: um loop. Executa o mesmo bloco de código durante um determinado número de vezes.

Cria uma variável de número inteiro que será incrementada a cada execução do laço.

Recebe o número inicial da variável e o número final que ela deve atingir.

```
for (i between 0, 10)
{
    ...
}

int valor_inicial = 5;
int valor_final = 20;

for (j between valor_inicial, valor_final)
{
    ...
}
```

Obs: as variáveis criadas no laço obedecem ao mesmo escopo que o restante do programa. Portanto, não é possível criar outra variável `i` no exemplo acima.

Programas de Exemplo

Junto ao projeto, na pasta `examples`, é possível encontrar alguns exemplos de programas válidos para Moe.

Como utilizar

Moe foi criada com as ferramentas Flex e Bison. Portanto, é necessário possuí-las instaladas caso queira buildar o projeto, além da ferramenta Make.

Uma vez com essas ferramentas, basta criar o diretório bin na mesma pasta do projeto e executar o seguinte comando no terminal:

```
make build
```

Ao final da execução, terá sido gerado um executável chamado moe.exe na pasta bin.

Executando este programa no terminal (exemplo: .\bin\moe.exe), ele passará a esperar a entrada de um programa Moe válido.

Se inserido um programa válido, ele irá gerar uma saída em formato ACL (a.txt) na pasta onde o programa foi chamado.

```
.\bin\moe.exe
program P { int x = 0; }
```

Também é possível passar programas como argumento:

```
.\bin\moe.exe .\examples\example1.moe
```

Bonus

Na pasta bonus, há um esboço do que seria o mesmo compilador refeito em C#.

Algumas estruturas principais, como o Lexer e o Parser, estão disponíveis, mas análise semântica e o reconhecimento de erros ainda estavam começando a ser construídas.

Infelizmente, não foi possível concluído dentro do prazo.

Grupo

[081200016] Carlos Eduardo Vieira Santos
[081200007] Guilherme Dias Lima Turtera
[081200011] Caio Rodrigues Fernandes Santos
[081200028] Nathan Vilela de Souza
[081200037] Gabriel Mendes Rodrigues Oliveira

Referências

O projeto contou com apoio das seguintes referências:

- Manual do ACL

Moe:
- Building a C Compiler using Lex and Yacc, por Anjaneya Tripathi
- Lex and YACC primer/HOWTO
- Exemplo de analisador léxico utilizando Lex, por westes
- Série em duas partes sobre Lex e YACC
- Lex and YACC program information, IBM

CLACL: C-Like ACL (Moe em C#):
- Crafting Interpreters, por Robert Nystrom