ESQUEMA DE TRADUÇÃO
(para implementação do analisador semântico e gerador de código)

```
<programa>              ::= #1 main module [ <lista módulos> #2 <lista variáveis> <lista comandos> ] #3
<lista módulos>         ::= ε | <módulo> <lista módulos>
<módulo>                ::= module <identificador> #4 <parâmetros> #5 [
                                <lista variáveis> <lista_cmd módulo> return <expressão> ; ] #6
<parâmetros>            ::= ε | : <lista_id> #7
<lista_cmd módulo>      ::= ε | <lista comandos>
<lista variáveis>       ::= ε | : <lista_id> #8 ; <lista variáveis>
<lista_id>              ::= <identificador> #9 | <identificador> #9 , <lista_id>
<identificador>         ::= id_int | id_float | id_string | id_boolean
<lista comandos>        ::= <comando> | <comando> <lista comandos>
<comando>               ::= <atribuição> | <entrada> | <saída> | <seleção> | <repetição>
<atribuição>            ::= <identificador> #9 <- <expressão> #10 ;
<entrada>               ::= in ( <lista_id > ) #11 ;
<saída>                 ::= out (#12 <lista expressões> ) ;
<lista expressões>      ::= <expressão> #13 | <expressão> #13 , <lista expressões>
<seleção>               ::= if ( <expressão> #14 ) isTrueDo : [ <lista comandos> ] <isFalseDo> #15
<isFalseDo>             ::= ε | #16 isFalseDo : [ <lista comandos> ]
<repetição>             := while #17 ( <expressão> ) isTrueDo  #18 : [ <lista comando> ] #19
                        |  while #17 ( <expressão> ) isFalseDo #18 : [ <lista comando> ] #19
<expressão>             ::= <expressão> or <valor> #20 | <expressão> and <valor> #21 | <valor>
<valor>                 ::= <relacional> | true #22 | false #23 | not <valor> #24
<relacional>            ::= <aritmética> <operador relacional> #25 <aritmética> #26 | <aritmética>
<operador relacional>   ::= = | != | < | <= | > | >=
<aritmética>            ::= <aritmética> + <termo> #27 | <aritmética> - <termo> #28 | <termo>
<termo>                 ::= <termo> * <fator> #29 | <termo> / <fator> #30 | <fator>
<fator>                 ::= <identificador> #9 #31
                          | <identificador> #9 (#32 <lista expressão> )  #33
                          | constante_int #34
                          | constante_float #35
                          | constante_literal #36
                          | ( <expressão> )
                          | + <fator> #37
                          | - <fator> #38
```

DESCRIÇÃO DOS REGISTROS SEMÂNTICOS: para executar a análise semântica e a geração de código é necessário fazer uso de registros semânticos (outros podem e devem ser definidos, bem como os descritos abaixo podem ser alterados, conforme a implementação das ações semânticas):

- **`operador_relacional`** (inicialmente igual a `''`): usado para armazenar o operador relacional reconhecido pela **ação #25**, para uso posterior na **ação #26**.
- **`código`**: usado para armazenar o código objeto gerado.
- **`tabela_de_símbolos`** (inicialmente vazia): usada para armazenar informações sobre os identificadores declarados. Cada linha da tabela tem quatro campos:

identificador	tipo em MSIL	qdade de parâmetros	tipos dos parâmetros em MSIL
de variável **`int`**	**`int64`**	–	–
de variável de **`float`**	**`float64`**	–	–
de variável de **`string`**	**`string`**	–	–
de variável de **`bool`**	**`bool`**	–	–
de **`module`**	**`<tipo>`**, onde **`<tipo>`** pode ser **`int64`**, **`float64`**, **`string`** ou **`bool`**, dependendo do identificador do **`module`**	constante inteira	lista de **`<tipo>`**s, onde **`<tipo>`** pode ser **`int64`**, **`float64`**, **`string`** ou **`bool`**, dependendo do identificador do parâmetro
de parâmetro **`int`**	**`int64`**	–	–
de parâmetro **`float`**	**`float64`**	–	–
de parâmetro **`string`**	**`string`**	–	–
de parâmetro **`bool`**	**`bool`**	–	–

Observa-se que a coluna `tipo em MSIL` não é necessária, uma vez que é possível identificar o tipo do `identificador` pelo padrão de formação do mesmo (informação contida na especificação léxica). Assim também não é necessária a coluna `qdade de parâmetros`, uma vez que a quantidade de parâmetros do módulo (**`module`**) pode ser definida pelo número de elementos da `lista de` **`<tipo>`**`s` dos parâmetros do módulo. Para determinar o escopo de um `identificador` pode ser incluída outra coluna (`escopo`) na **`tabela_de_símbolos`**. Outra alternativa é criar

uma nova **tabela_de_símbolos** no início da compilação de um módulo e destruí-la no final, para armazenar os parâmetros e as variáveis locais ao módulo.

- **lista_de_identificadores** (inicialmente vazia): usada para armazenar os identificadores reconhecidos pela **ação #9**, para uso posterior em outras ações semânticas.
- **pilha_de_rótulos** (inicialmente vazia): usada na análise dos comandos de seleção e de repetição.
- **pilha_de_tipos** (inicialmente vazia): usada para determinar o tipo de uma expressão.

DESCRIÇÃO DAS VERIFICAÇÕES SEMÂNTICAS:

- ✓ A linguagem NÃO é *case sensitive*.
- ✓ O escopo dos identificadores é conforme segue:
  - o identificador de um módulo (**module)** deve ser visível a todos os demais módulos e aos comandos do módulo principal (**main module**);
  - os parâmetros e as variáveis locais aos módulos são visíveis apenas no próprio módulo;
  - as variáveis do módulo principal são visíveis aos comandos do módulo principal.
- ✓ Qualquer identificador só pode ser declarado uma vez no escopo correspondente.
- ✓ Qualquer identificador só pode ser usado se for declarado no escopo correspondente.
- ✓ O tipo de uma <expressão> deve ser determinado da seguinte forma:

tipo dos operandos	operadores	tipo da expressão resultante
constante_int		**int64**
constante_float		**float64**
constante_literal		**string**
true ou **false**		**bool**
<identificador>		**int64** ou **float64** ou **string** ou **bool** conforme declaração
<identificador> **(** <lista expressão> **)**		**int64** ou **float64** ou **string** ou **bool** conforme declaração //retorna o valor resultante da execução do //módulo
int64, int64	**+ - * / sinais unários**	**int64**
float64, float64	**+ - * / sinais unários**	**float64**
int64, int64	**= != < <= > >=**	**bool**
float64, float64	**= != < <= > >=**	**bool**
string, string	**= != < <= > >=**	**bool**
bool, bool	**and or not**	**bool**

Operadores e tipos não previstos na tabela anterior indicam que a operação correspondente não pode ser executada com os tipos em questão.

- ✓ Quanto à compatibilidade de tipos, tem-se as seguintes regras:
  - no comando <atribuição>, variáveis do tipo **int64** só podem armazenar valores do tipo **int64**; variáveis do tipo **float64** só podem armazenar valores do tipo **float64**; variáveis do tipo **string** só podem armazenar valores do tipo **string**; variáveis do tipo **bool** só podem armazenar valores do tipo **bool**;
  - no comando <entrada>, os identificadores não podem ser do tipo **bool** ou de módulo;
  - nos comandos <seleção>, a <expressão> deve ser do tipo **bool**;
  - no comando <repetição>, a <expressão> deve ser do tipo **bool**;
  - na declaração de um módulo, a <expressão> do comando **return** <expressão> deve ser do mesmo tipo do módulo;
  - na chamada de um módulo, a quantidade e os tipos dos argumentos devem ser iguais a quantidade e aos tipos dos parâmetros.

DESCRIÇÃO DA SEMÂNTICA:

- ✓ A semântica da declaração de variáveis (<lista variáveis>) é a seguinte: incluir cada identificador da <lista_id> na **tabela_de_símbolos** com o tipo e no escopo correspondente, conforme declarado; gerar código para alocar memória para o(s) identificador(es) declarado(s).
- ✓ A semântica do comando <atribuição> é a seguinte: gerar código, conforme descrito abaixo, para avaliar a <expressão>; gerar código para atribuir o resultado da avaliação da <expressão> ao identificador.
- ✓ A semântica do comando <entrada> é a seguinte: para cada identificador da <lista_id>, gerar código para ler (da entrada padrão) um valor; gerar código para armazenar o valor lido no identificador correspondente.

- ✓ A semântica do comando <saída> é a seguinte: para cada <expressão> da <lista expressões>, gerar código para escrever (na saída padrão) o resultado da avaliação da <expressão>, sem quebra de linha.
- ✓ A semântica do comando <seleção> é a seguinte: gerar código para verificar se o resultado da avaliação da <expressão> é verdadeiro. Em caso positivo, gerar código para executar apenas os comandos da <lista comandos> associada à cláusula **isTrueDo**. Em caso negativo, gerar código para executar apenas os comandos da <lista comandos> da cláusula **isFalseDo**, se existir.
- ✓ A semântica do comando <repetição> **while-isTrueDo** é a seguinte: gerar código para verificar se o resultado da avaliação da <expressão> é verdadeiro. Em caso positivo, gerar código para executar os comandos da <lista comandos>, gerar código para repetir a execução. Em caso negativo, gerar código para executar o primeiro comando após o comando de repetição.
- ✓ A semântica do comando <repetição> **while-isFalse** é a seguinte: gerar código para verificar se o resultado da avaliação da <expressão> é falso. Em caso positivo, gerar código para executar os comandos da <lista comandos>, gerar código para repetir a execução. Em caso negativo, gerar código para executar o primeiro comando após o comando de repetição.
- ✓ A semântica de uma expressão (<expressão>) é a seguinte: para <identificador>, se for identificador de variável ou de parâmetro, gerar código para carregar o valor do identificador, e se for identificador de módulo, gerar código para efetuar a chamado do módulo com os argumentos correspondentes; para constantes (constante_int, constante_float, constante_literal, **true**, **false**), gerar código para carregar o valor da constante; para os operadores (lógicos, relacionais, aritméticos), gerar código para efetuar a operação correspondente.
- ✓ A semântica da declaração de um módulo (<módulo>) é a seguinte: incluir o identificador do <módulo> na **tabela_de_símbolos** com o tipo, o número de parâmetros, os tipos dos parâmetros e no escopo correspondente, conforme declarado; incluir cada identificador da lista de <parâmetros> na **tabela_de_símbolos** com o tipo e no escopo correspondente, conforme declarado; gerar código para declarar os parâmetros; gerar código para retornar o resultado da avaliação da <expressão> (em **return** <expressão>). A semântica da declaração de variáveis (<lista variáveis>) e dos comandos (<lista_cmd módulo>) foi descrita anteriormente.

SUGESTÃO DE IMPLEMENTAÇÃO DAS AÇÕES SEMÂNTICAS (por grau de dificuldade, grupo de ações):

GRAU	AÇÃO SEMÂNTICA	DESCRIÇÃO
1 (mais fácil)	1, 2, 3, 12, 13, 20-30, 34-38 38, 39	
2	8, 9, 10, 11, 31	ações que usam identificadores
3	14-19	ações para comandos de seleção e repetição
4	4, 5, 6, 7, 32, 33	ações para módulo