Universidade Estadual de Londrina

DC Departamento de Computação

5COP093 - Compilador C: Analisador Sintático C Simplificado

Utilizando a ferramenta Bison, implemente um analisador sintático para o **subconjunto** da linguagem C apresentado nos diagramas sintáticos presentes neste texto. Para a análise léxica, utilize o analisador léxico desenvolvido em Flex na etapa anterior do compilador, realizando as devidas adequações para que o mesmo seja utilizado pelo *parser* gerado pelo Bison.

Nesta etapa, o analisador sintático C deve reconhecer somente os *tokens* que são apontados como símbolos terminais nos diagramas sintáticos C. Os demais *tokens*, mesmo que reconhecidos pelo analisador léxico devem ser apontados como erro sintático como, por exemplo, o *token* `struct`, o qual é reconhecido pelo léxico mas que deve ser recusado pelo sintático.

Neste ponto do trabalho, o compilador dever ser capaz de reconhecer erros léxicos e sintáticos, sendo que agora o processo de compilação deve terminar quando o primeiro erro (léxico ou sintático) for encontrado.

Quando um erro léxico for detectado, deve-se mostrar a linha e a coluna onde o erro ocorreu.

Exemplo de entrada:

```
void f(){ if(@)
{
   a
}
```

Saída esperada:

```
error:lexical:1:14: @
```

Observe que na saída esperada, a mensagem de erro apresentada foi:

```
error:lexical:1:14: @
```

onde o primeiro número indica a linha onde o erro ocorreu e o segundo número indica a coluna onde o erro ocorreu. No exemplo dado, o erro ocorreu na linha 1, coluna 14. Observe também que deve ser impresso o caractere que causou o erro léxico. Observe que não é mais necessário imprimir os tokens, mas somente a mensagem de erro.

Observe o exemplo a seguir:

```
void f(){ if(1)
{
/* isto eh um
comentario iniciado
e nao terminado
```

Saída esperada:

```
error:lexical:3:1: unterminated comment
```

No exemplo apresentado, o arquivo termina com um comentário de bloco que não foi fechado. Tal tipo de erro deve ser apresentado com a mensagem padrão do exemplo. Observe também que mesmo o arquivo contendo 3 linhas de comentário de bloco não terminado, a mensagem de erro irá mostrar a linha onde o comentário se inicia. No caso do exemplo, o comentário se inicia na linha 3. Mesmo que o comentário possua inúmeras linhas, o erro deve apontar a linha onde o comentário é iniciado. Tal como outros erros léxicos, também deve-se informar a coluna onde o erro ocorreu. No exemplo apresentado, o erro ocorreu na coluna 1.

Observe o exemplo a seguir:

```
void f()
{
    if(struct)
    {
        main();
    }
}
```

Saída esperada:

```
error:syntax:3:8: struct
    if(struct)
       ^
```

No exemplo apresentado ocorreu um erro sintático na linha 3, coluna 8. Observe que o *token* causador do erro também é impresso, no caso do exemplo, o *token* `struct`. Outra particularidade desta mensagem de erro é que na linha seguinte é impresso a linha de código onde ocorreu o erro sintático, sendo que na terceira linha da mensagem de erro o símbolo `^` é impresso indicando a coluna 8, ou seja, o ponto exato que causou o erro sintático.

Caso nenhum erro léxico ou sintático seja encontrado, o compilador deve imprimir a mensagem:

```
SUCCESSFUL COMPILATION.
```

Observe o exemplo a seguir:

```
void f()
{
    if(1)
    {
        main();
    }
}
```

Saída esperada:

```
SUCCESSFUL COMPILATION.
```

Observe o exemplo a seguir:

```
int B,Q; void f() { /*comentario*/
```

Saída esperada:

```
error:syntax:1:35: expected declaration or statement at end of input
int B,Q; void f() { /*comentario*/
                                  ^
```

No exemplo apresentado não existe, em princípio, erro sintático, pois trata-se na verdade de um programa que foi iniciado e não terminado. Quando tal tipo de programa for compilado, deve-se exibir a mensagem `expected declaration or statement at end of input` ao invés de se apresentar um *token*. O programa do exemplo possui uma linha e 34 caracteres. Como deveria haver uma declaração ou comando após o último caractere, a coluna apresentada é a posterior ao último caractere do programa. Neste caso como o último caractere do programa fica na coluna 34, a mensagem de erro deve indicar a coluna 35. Observe que o símbolo ^ aponta para a coluna 35 ao imprimir a linha que causou o erro.

Observe o exemplo a seguir:

```
void f() { //comentario de linha
```

Saída esperada:

```
error:syntax:1:12: expected declaration or statement at end of input
void f() { //comentario de linha
           ^
```

Este exemplo também trata-se de um programa iniciado e não terminado que, além deste fato, não apresenta erro sintático anterior. O programa possui apenas uma única linha.

Da mesma forma que no exemplo anterior, deve-se exibir a mensagem `expected declaration or statement at end of input` ao invés de se apresentar um *token*.

Como o programa foi terminado com um comentário de linha, a declaração ou comando que deveria haver, deve aparecer antes do comentário de linha. Desta forma na mensagem de erro sintático, a coluna a ser indicada é a coluna onde se inicia o comentário de linha.

Recomendações

Por favor, evite escrever código da seguinte forma:

```
for(int i = 0; ...)
{
    ...
}
```

onde uma variável local está sendo declarada dentro de um comando. Se ainda sim quiser utilizar tal estilo de programação, não se esqueça de colocar no `Makefile` as devidas opções para que o compilador aceite tal construção, pois nem todos os compiladores a aceitam por padrão.

Em geral a opção `-std=c99` é o suficiente para que tal construção seja aceita e compilada sem maiores problemas. Sua utilização, em geral, é da seguinte forma:

```
$gcc -std=c99 teste.c -o teste
```

Se você programar utilizando `C++ 11`, por favor, utilize também a opção adequada do compilador para habilitar a compilação de tal versão do `C++`.

IMPORTANTE: Se ficou com alguma dúvida em relação a qualquer item deste texto, não hesite em falar com o professor da disciplina, pois ele está à disposição para sanar eventuais dúvidas, além do que, isso faz parte do trabalho dele.

Especificações de Entrega

O trabalho deve ser entregue no *AVA* em um arquivo .zip com o nome `sintatico.zip`. Este arquivo .zip deve conter somente os arquivos necessários à compilação, sendo que deve haver um `Makefile` para a geração do executável.

A entrega deve ser feita exclusivamente no *AVA* até a data/hora especificada. Não serão aceitas entregas atrasadas ou por outro meio que não seja o *AVA*.

Observação: o arquivo .zip não deve conter pastas, para que quando descompactado, os fontes do trabalho apareçam no mesmo diretório do .zip. O nome do executável gerado pelo `Makefile` deve ser `sintatico`.

O programa gerado deve ler as suas entradas da entrada padrão do sistema e imprimir as saídas na saída padrão do sistema. Um exemplo de execução para uma entrada chamada `teste.c` seria a seguinte:

```
$./sintatico < teste.c
```

Se o programa que for fornecido como entrada estiver correto, a seguinte mensagem deve ser impressa:

```
SUCCESSFUL COMPILATION.
```

Nenhuma linha extra deve ser gerada após a mensagem, ou seja, essa linha seria gerada pelo comando:

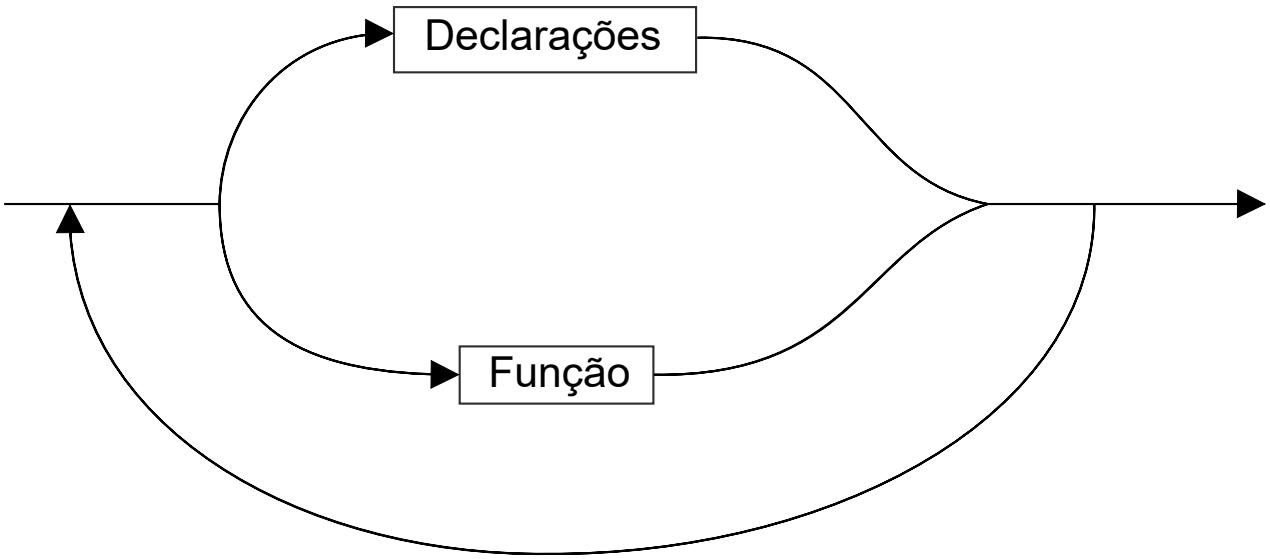
```
printf("SUCCESSFUL COMPILATION.");
```

Para erros léxicos e sintáticos também não devem haver linhas extras na mensagem de erro gerada.

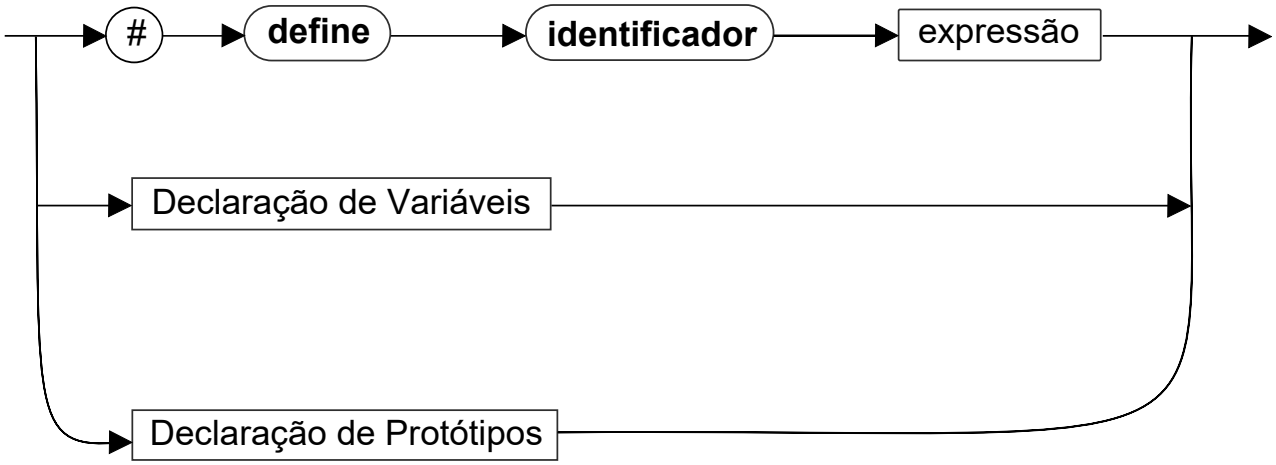
IMPORTANTE: Arquivos ou programas entregues fora do padrão receberão nota `ZERO`. Entende-se como arquivo fora do padrão aquele que tenha um nome diferente de `sintatico.zip`, que contenha subpastas ou não seja um `.zip`, por exemplo. Entende-se como programa fora do padrão aquele que não contiver um `Makefile`, que apresentar erro de compilação, que não ler da entrada padrão, não imprimir na saída padrão ou o nome do executável for diferente de `sintatico`, por exemplo. Uma forma de verificar se seu arquivo ou programa está dentro das especificações é testar o mesmo com o `script` de testes que é fornecido no *AVA*. Se o seu arquivo/programa **não** funcionar com o `script`, significa que ele está **fora** das especificações e, portanto, receberá nota **ZERO**.

Diagramas Sintáticos C

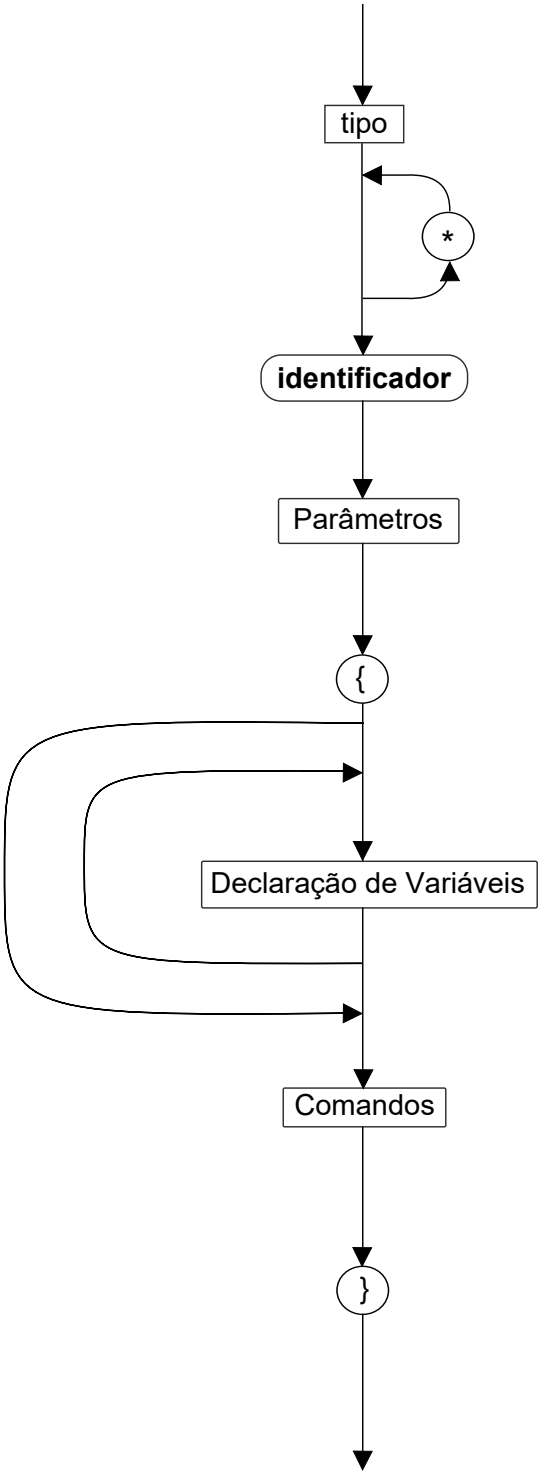
Programa:



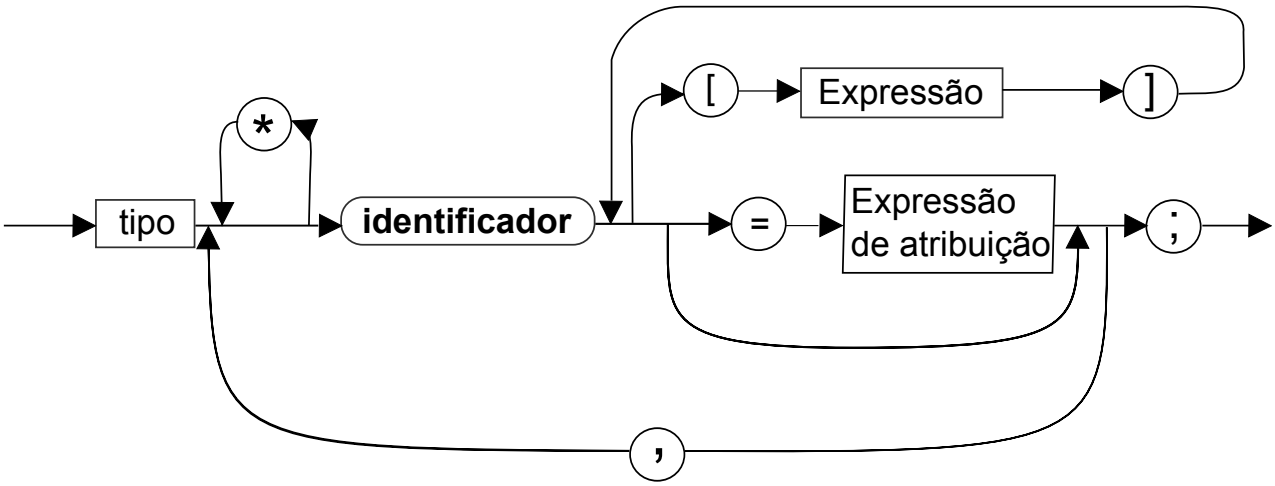
Declarações:



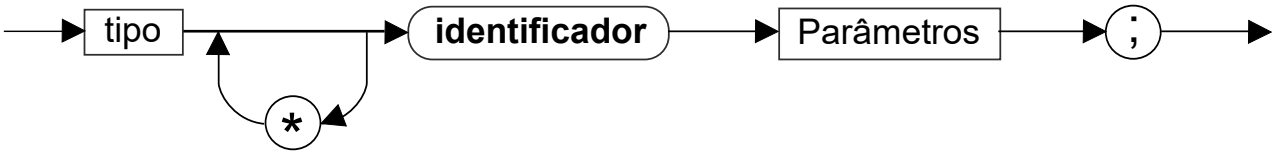
Função:


tipo
*
identificador
Parâmetros
{
Declaração de Variáveis
Comandos
}

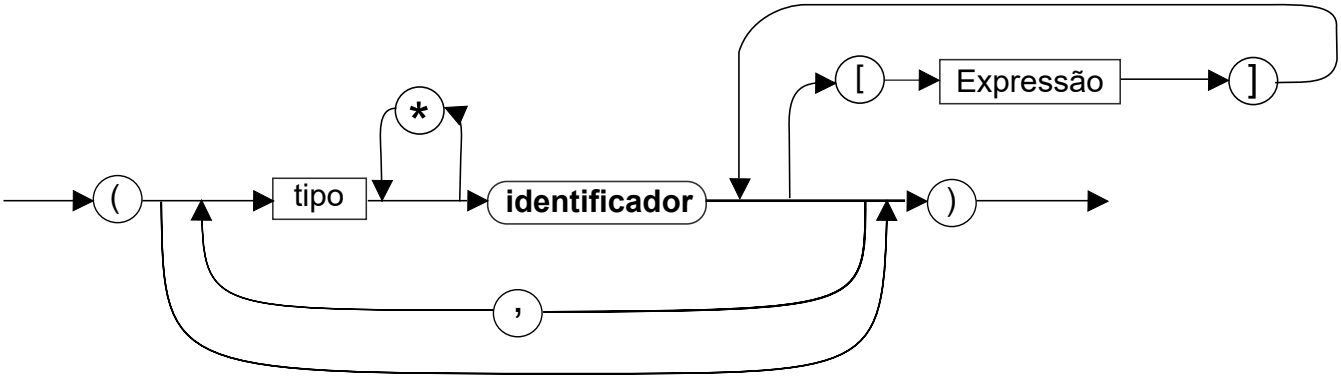
Declaração de Variáveis:



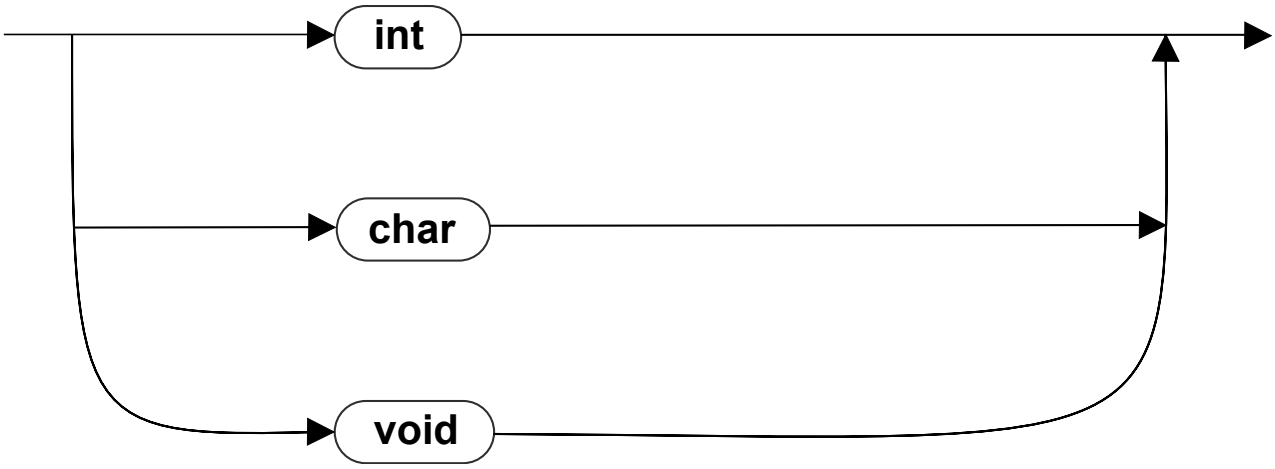
Declaração de Protótipos:



Parâmetros:

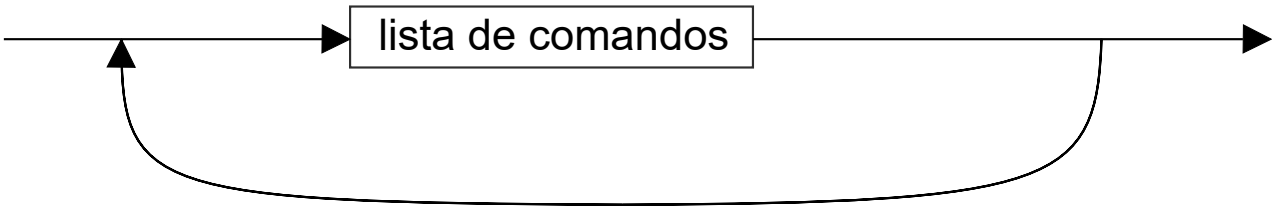


Tipo:

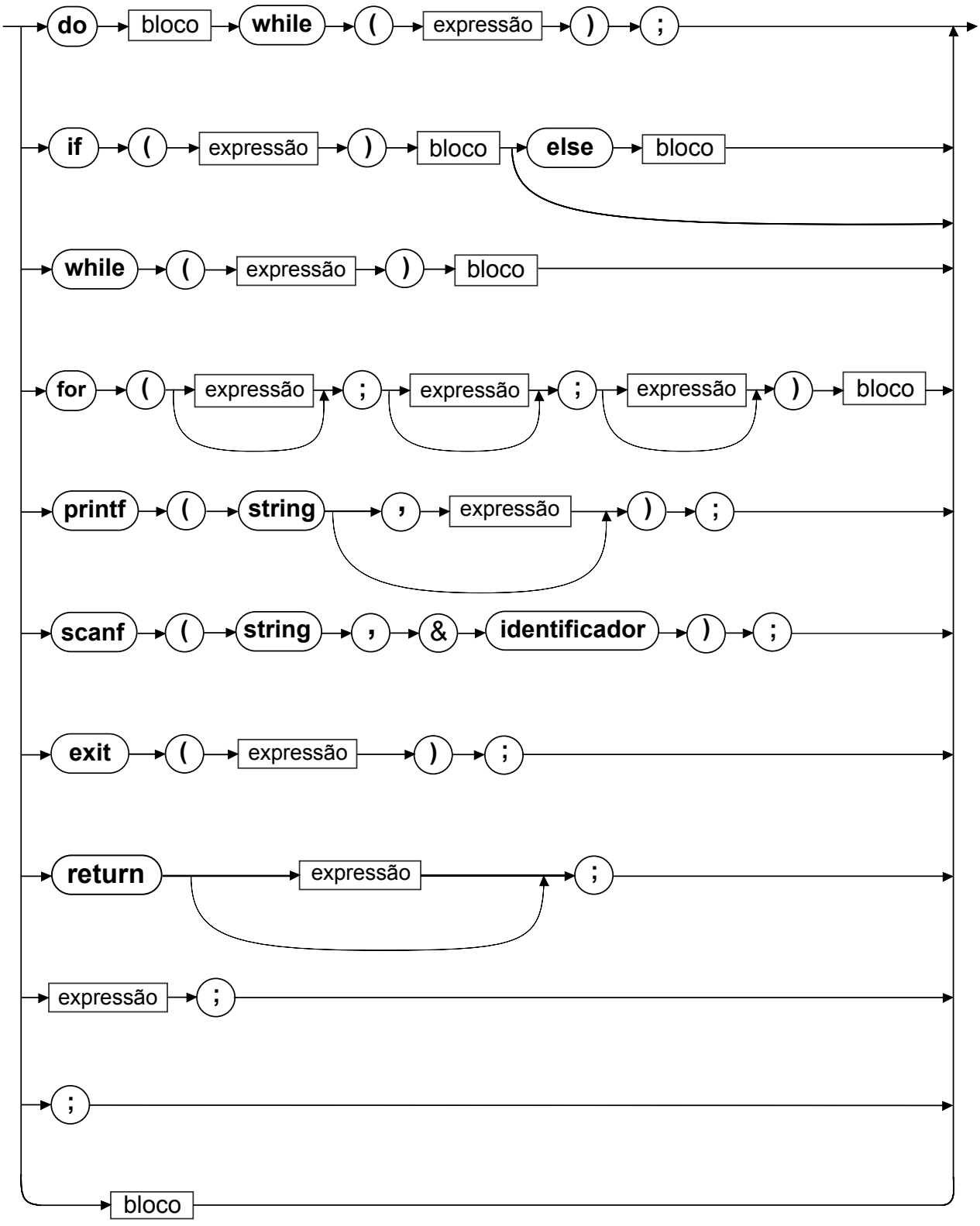


Bloco:

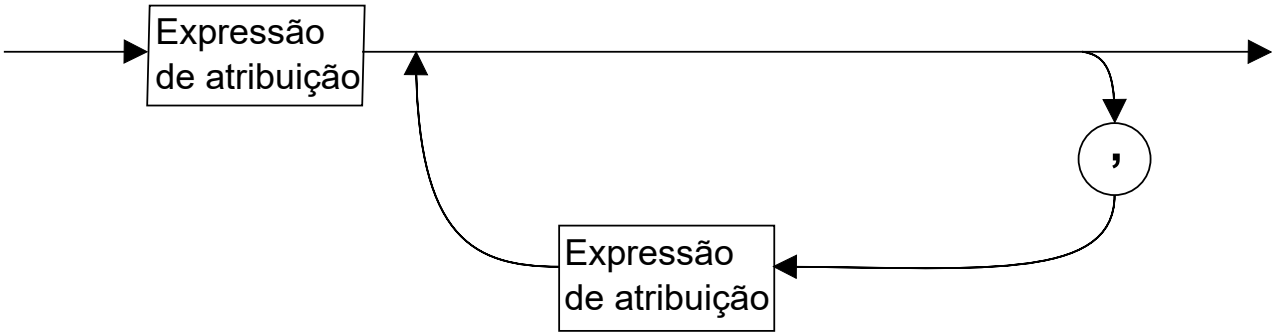
Comandos:



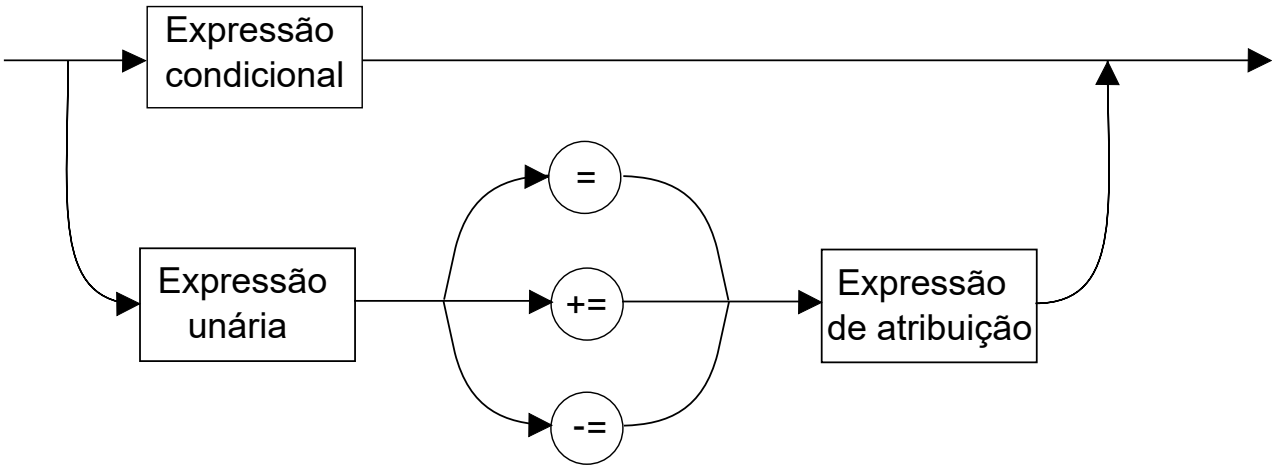
Lista de comandos:



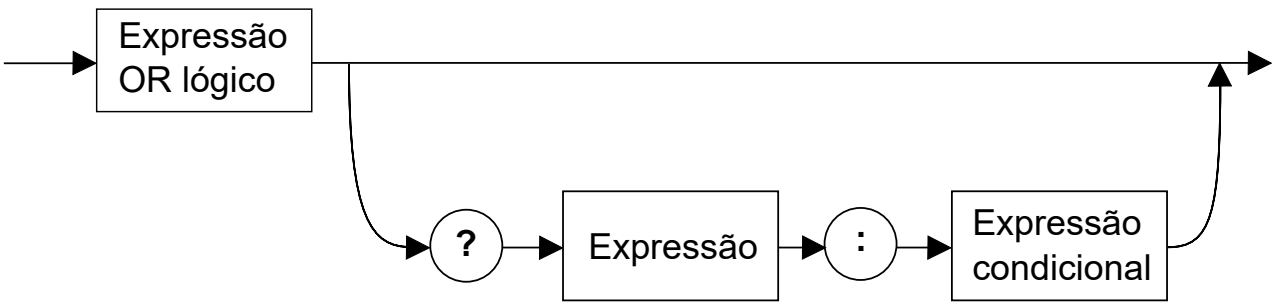
Expressão:



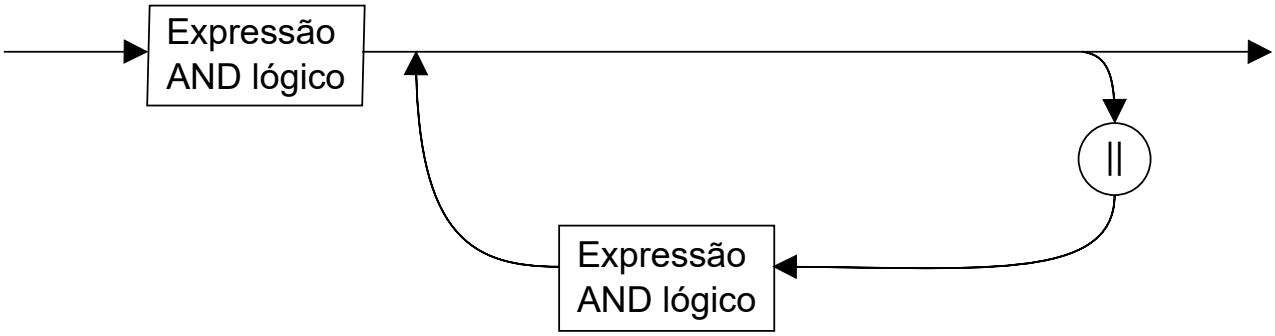
Expressão de Atribuição:



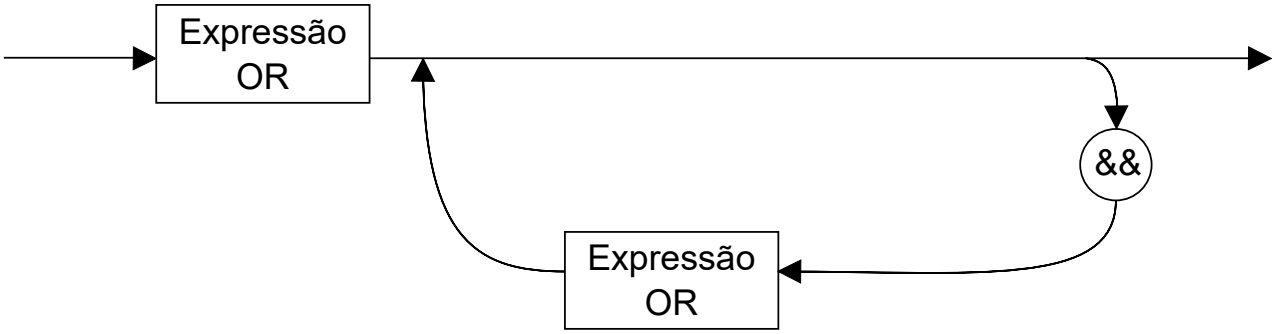
Expressão Condicional:



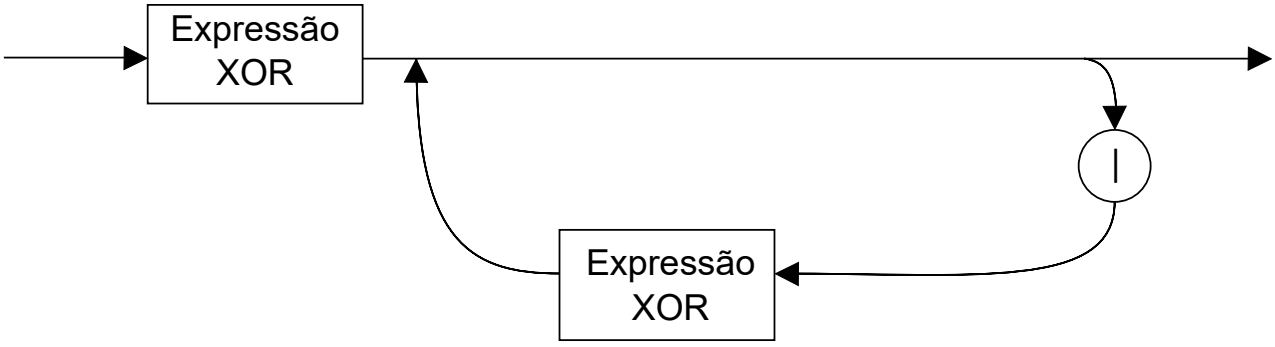
Expressão OR lógico:



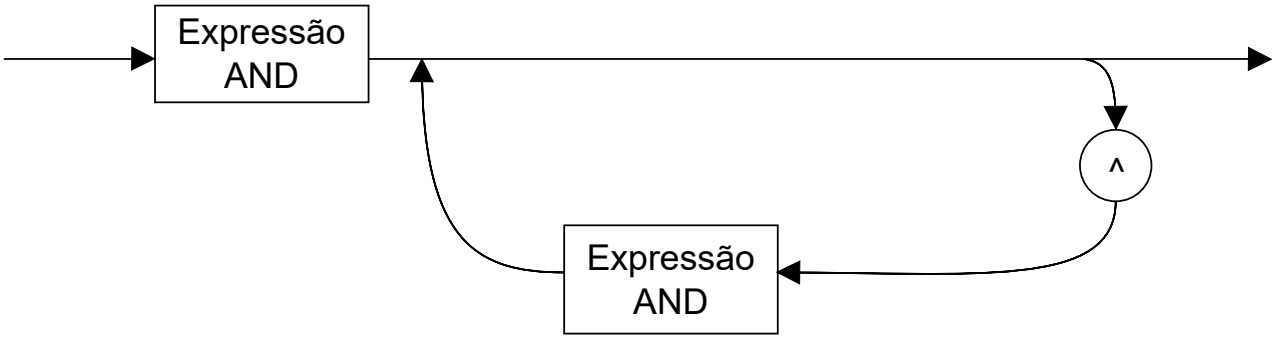
Expressão AND lógico:



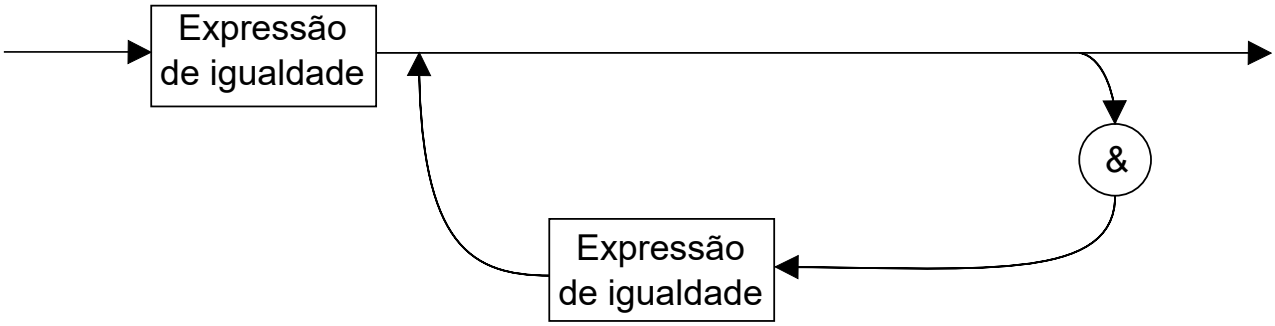
Expressão OR:



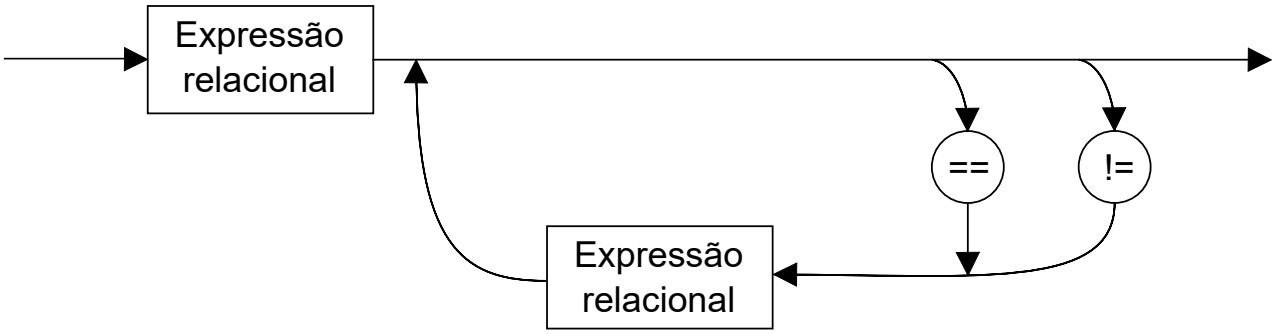
Expressão XOR:



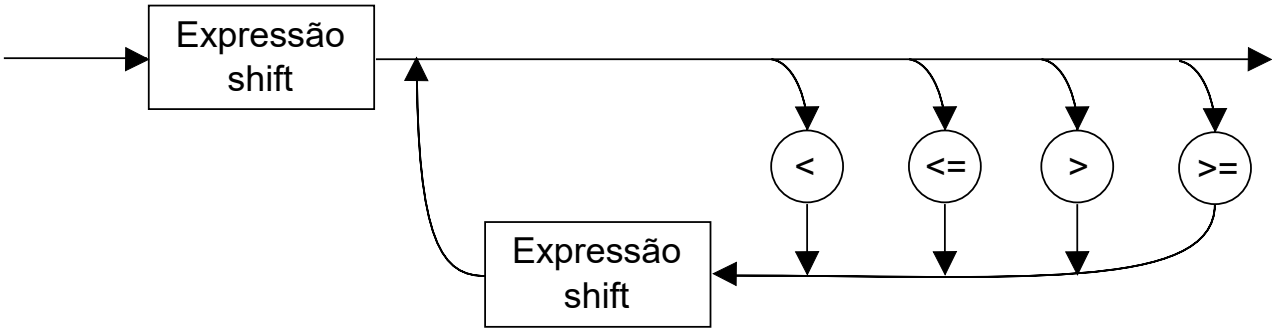
Expressão AND:



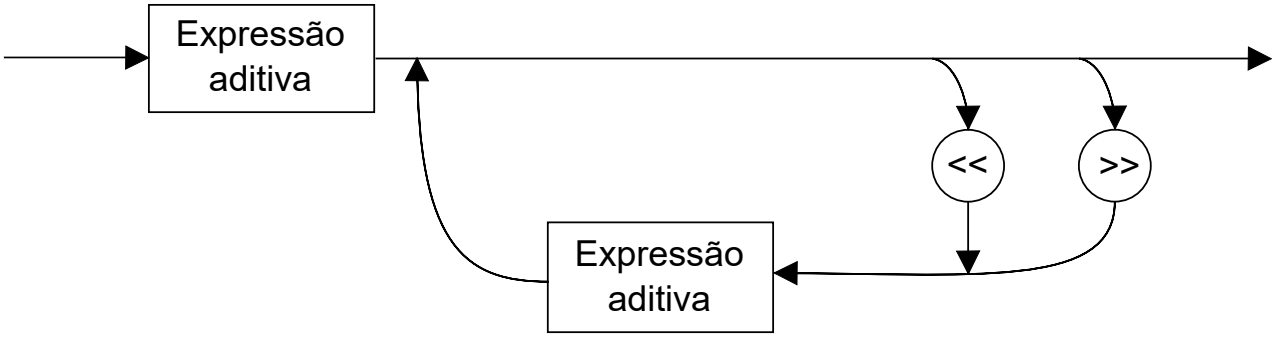
Expressão de Igualdade:



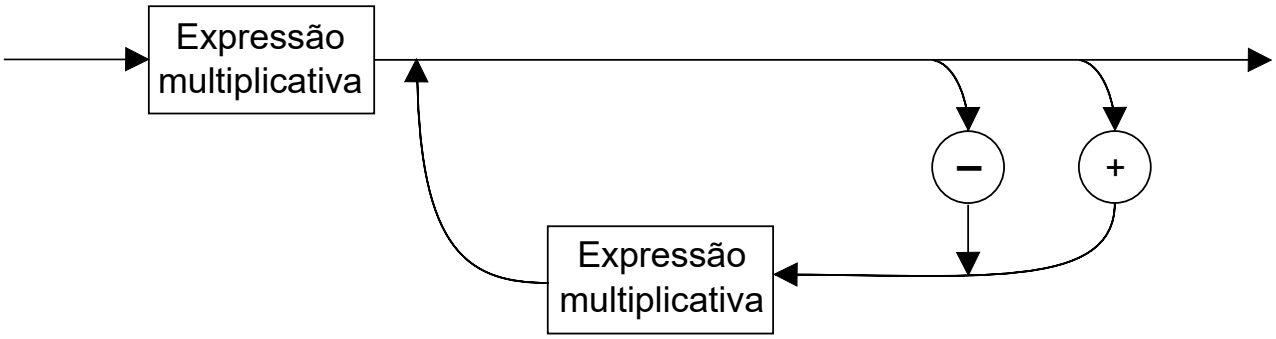
Expressão Relacional:



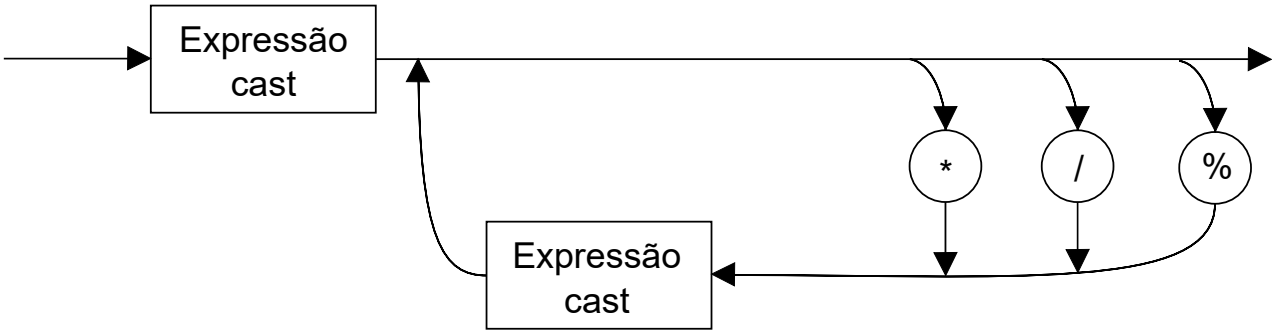
Expressão Shift:



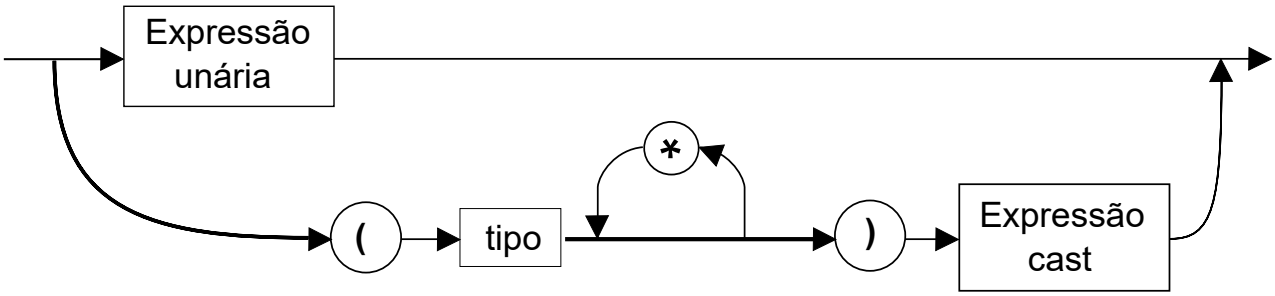
Expressão aditiva:



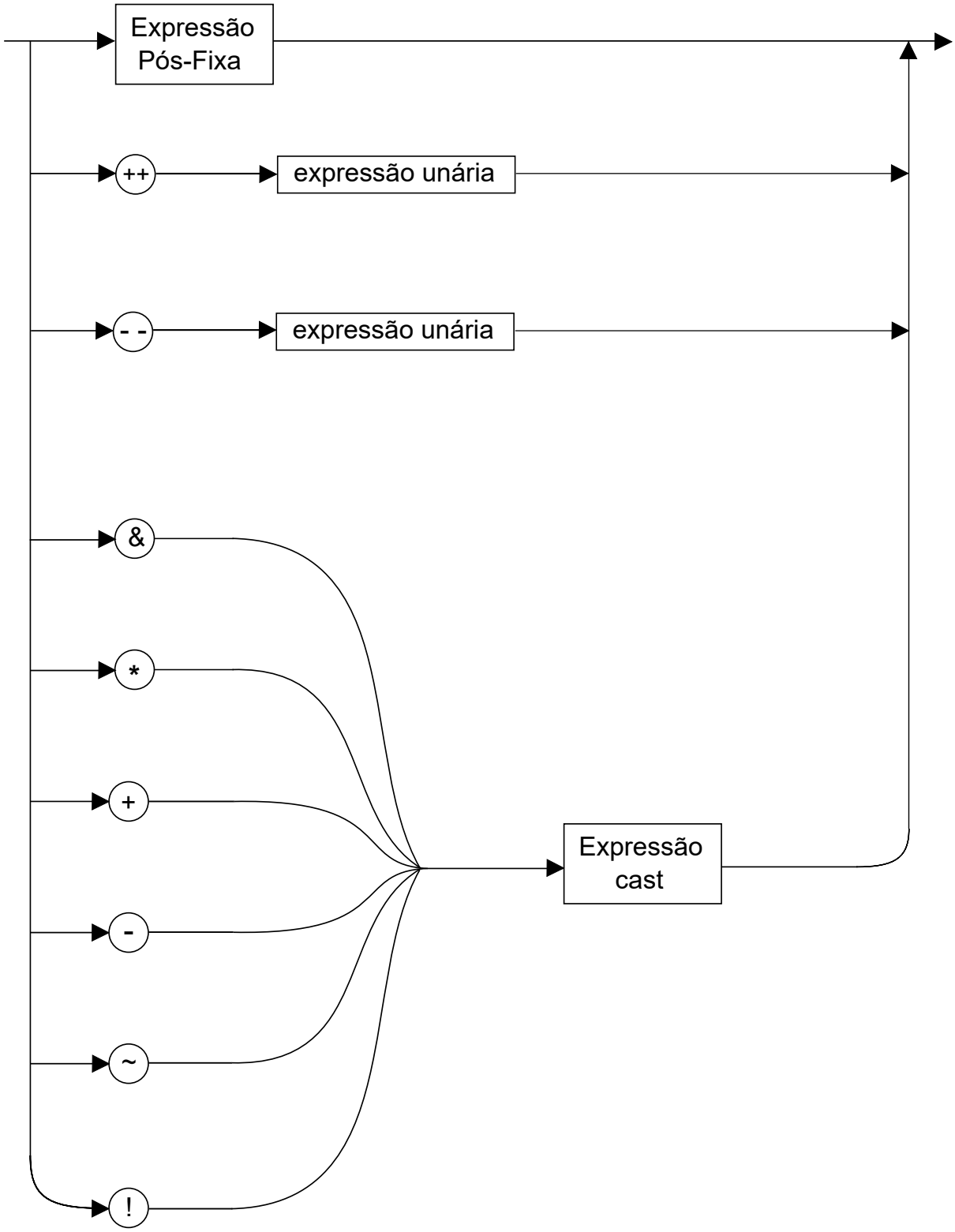
Expressão multiplicativa:



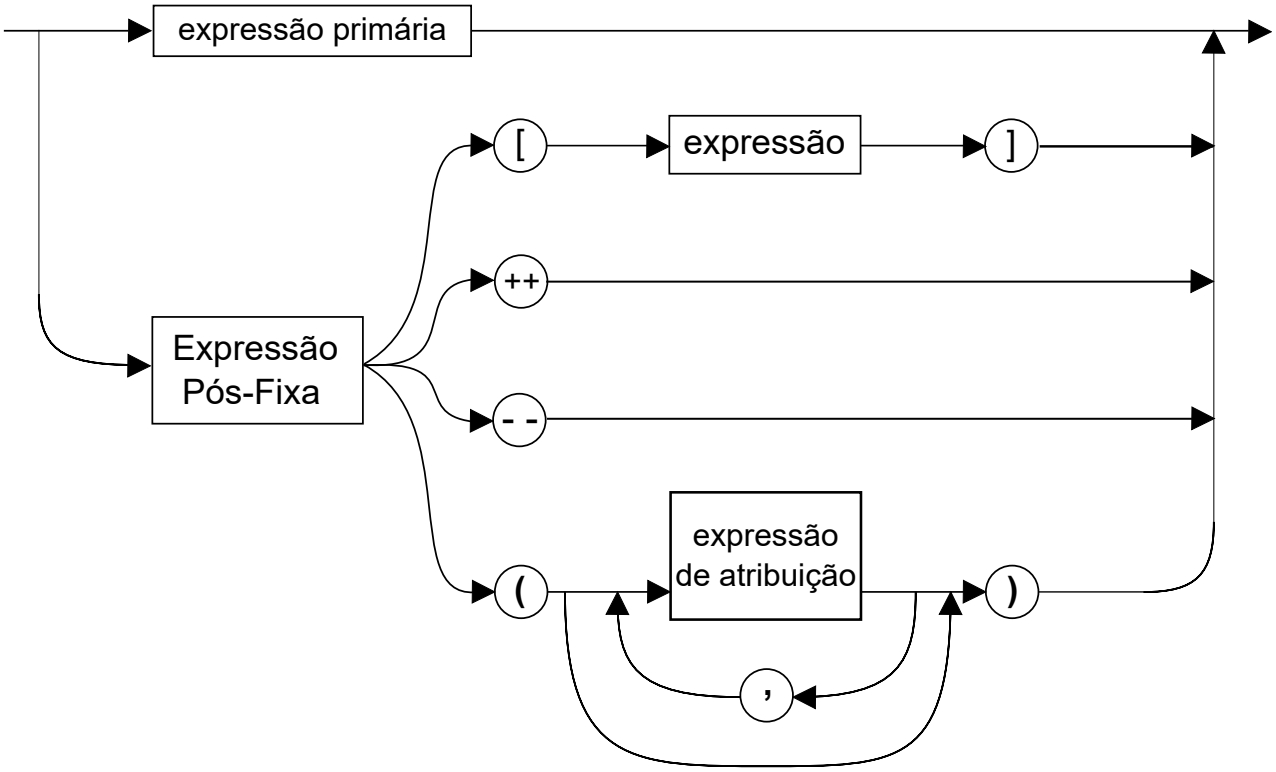
Expressao Cast:



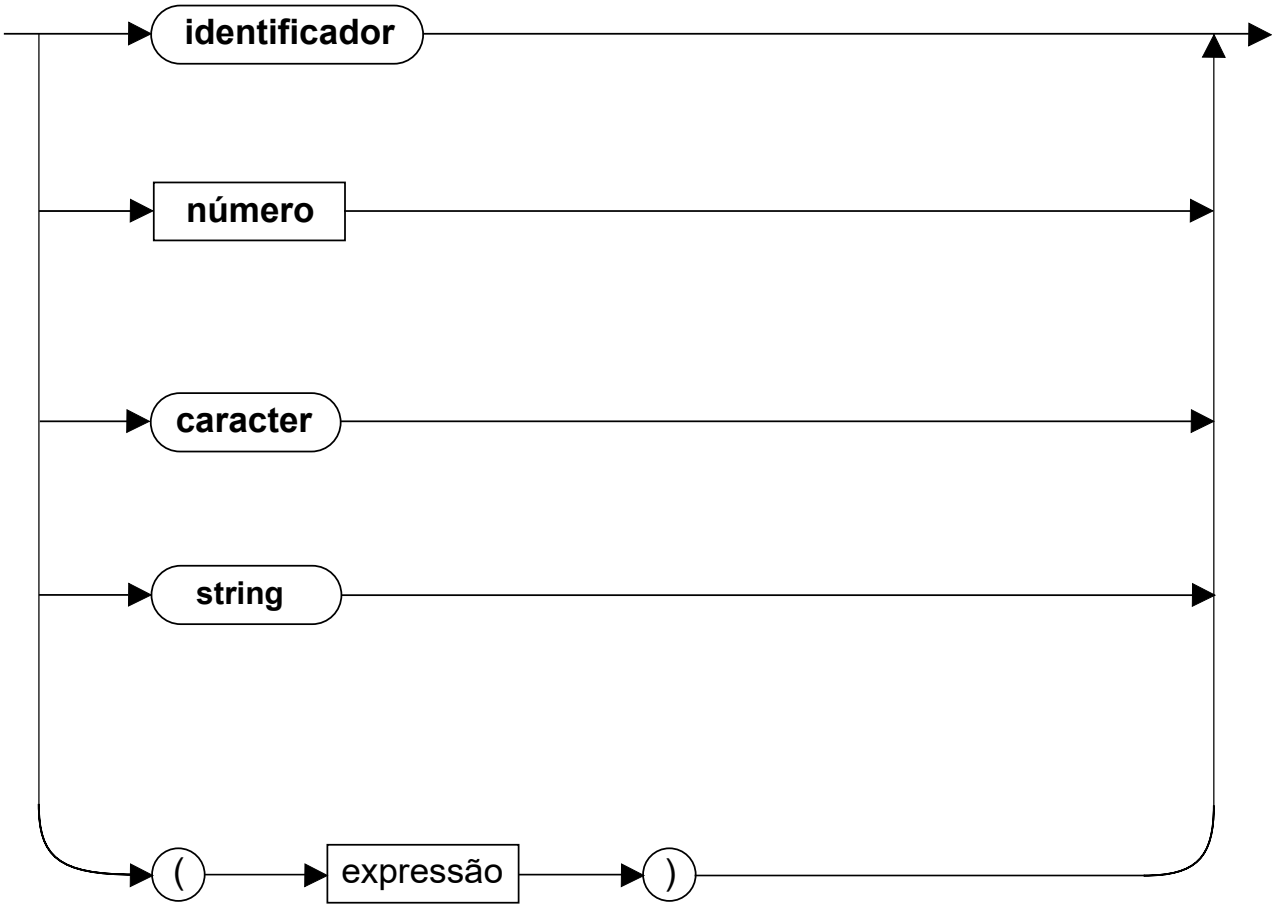
Expressão unária:



Expressão Pós-Fixa:



Expressão Primária:



Número: