
G-Portugol

Manual da versão v1.1

Thiago Silva
tsilva@sourcecraft.info

20 de agosto de 2010

Sumário

1	Introdução	1
2	Características Gerais	2
2.1	Tipos de dados	2
2.2	Estruturas de controle	2
2.3	Subprogramas (funções)	2
3	Programando em G-Portugol	4
3.1	Olá Mundo	4
3.2	Variáveis	5
3.2.1	Variáveis primitivas	5
3.2.2	Vetores e matrizes (conjuntos)	6
3.3	Estruturas condicionais	6
3.4	Estruturas de repetição	8
3.4.1	A estrutura “enquanto”	8
3.4.2	A estrutura “repita”	8
3.4.3	A estrutura “para”	9
3.5	Funções	11
3.5.1	Funções internas	11
4	Implementação da linguagem G-Portugol	13
4.1	Introdução	13
4.2	A linguagem	13
4.2.1	Diretrizes para o design da linguagem	13
4.3	Formato Estrutural	15
4.3.1	Declaração do algoritmo	15
4.3.2	Declaração de variáveis globais	15
4.3.3	Bloco Principal	15
4.3.4	Atribuições	17
4.4	Funções	17
4.5	Funções internas	18
4.5.1	A função “imprima”	19
4.5.2	A função “leia”	19
5	O programa GPT	20
5.1	Introdução	20
5.2	Opções gerais	20
5.3	Tratamento de erros	21
5.4	Execução de programas	21
5.4.1	Compilação e geração de código executável	21
5.4.2	Tradução para a linguagem C	21
5.4.3	Interpretação de código	22
5.4.4	Processando algoritmos divididos em múltiplos arquivos	22

A Gramática da linguagem G-Portugol	23
A.1 Termos léxicos	23
A.2 Gramática	23

Resumo

Esse não é um livro que ensina programação, algoritmos ou lógica. Seu objetivo é servir de manual para a linguagem G-Portugol e ferramentas relacionadas. Portanto, ele assume que o leitor seja versado em linguagens de programação e desenvolvimento de software.

Capítulo 1

Introdução

G-Portugol é um dialeto da linguagem/pseudo-código portugol (ou português estruturado), que é muito usada para descrever algoritmos em português, de forma livre e espontânea. Em geral, livros dedicados ao ensino de algoritmos, lógica e estruturas de dados utilizam alguma forma dessa linguagem.

A proposta de G-Portugol é disponibilizar uma implementação da linguagem portugol, fornecendo ferramentas que ofereçam recursos de edição, compilação, execução e depuração de programas escritos nessa linguagem, de forma a favorecer aos estudantes que dão os primeiros passos no aprendizado de desenvolvimento de softwares, bem como professores que ensinam disciplinas relacionadas a computação. Portanto, seu foco é primariamente didático.

Encontram-se disponíveis atualmente um compilador, tradutor e interpretador para a linguagem (GPT) e um ambiente visual simples (GPTEditor) que permite a edição, execução e depuração de programas escritos em G-Portugol.

A seguir são apresentados os assuntos abordados nos capítulos seguintes:

- Capítulo 2: pretende discutir as características gerais da linguagem.
- Capítulo 3: aborda a programação em G-Portugol, suas estruturas e recursos, utilizando exemplos ilustrativos e comparando com linguagens populares. Embora o capítulo trate da programação, ele não tem como objetivo explicar programação em si ou a teoria/história por trás das estruturas abordadas.
- Capítulo 4: trata da implementação da linguagem. Discute as decisões de design e recursos oferecidos por G-Portugol.
- Capítulo 5: descreve o programa GPT.

No apêndice deste livro pode ser encontrado a gramática da linguagem G-Portugol.

Capítulo 2

Características Gerais

A linguagem em si não difere fundamentalmente ou apresenta novidades em relação ao uso popular de português. Semelhante às linguagens como Pascal e C, é uma linguagem imperativa, com comandos de controle de fluxo, manipulação de dados e recursos de entrada e saída básicos. A linguagem em *case sensitive*, o que significa que, por exemplo, uma função chamada “leia” é diferente de uma função chamada “Leia”.

2.1 Tipos de dados

Tipos primitivos como inteiro e literal são suportados. Literais são tipos integrais, e não, tipos construídos pelo usuário. A tabela 2.1 apresenta os tipos com respectivos exemplos de dados:

Tipos primitivos	
Tipo	Exemplos
inteiro	19
real	2.5
caractere	'a'
literal	“uma frase”
lógico	verdadeiro

Tabela 2.1: Tipos de dados primitivos

Com esses tipos, pode-se criar conjuntos como vetores ou matrizes “n” dimensionais. Tipos mais complexos não são suportados.

2.2 Estruturas de controle

O conjunto de estruturas de controle são os mais primitivos. Uma estrutura condicional (se/senão) e duas estruturas de repetição (enquanto/para) são suportadas. Embora a estrutura “para” seja uma especialização de um laço simples como o “enquanto”, ela foi implementada, visto que uma variedade de livros e muitos professores os discutem.

2.3 Subprogramas (funções)

Subprogramas são como funções em C. Podem receber qualquer número de parâmetros, sejam tipos primitivos ou vetores/matrizes, e podem retornar apenas valores primitivos. Entretanto, não é permitido declarar

Estruturas de repetição	
Estrutura	Tipo
se/então/senão/ enquanto repita para	condicional repetição repetição repetição

Tabela 2.2: Estruturas de repetição

funções aninhadas ou funções com parâmetros variáveis. Vale ressaltar que passagem de parâmetros é sempre feita por *valor*.

Capítulo 3

Programando em G-Portugol

A proposta desse capítulo é mostrar os elementos da linguagem G-Portugol usando programas como exemplos. Não é um capítulo sobre como programar, nem tem como intuito ensinar algoritmos. Portanto, é esperado que o leitor seja familiarizado com programação.

3.1 Olá Mundo

Mantendo a tradição, vamos criar nosso primeiro programa. O propósito dele será exibir na tela o texto "Olá mundo". Crie um arquivo chamado "olamundo.gpt" com o seguinte conteúdo:

Programa 1 "Olá Mundo" em G-Portugol.

```
/*
    Nosso primeiro programa
*/

algoritmo olamundo;

início
    imprima("Olá mundo!");
fim
```

Após salvar o arquivo, digite o seguinte na linha de comando:

No Linux:

```
$ gpt -o olamundo olamundo.gpt
```

No Windows:

```
\> gpt -o olamundo.exe olamundo.gpt
```

Esse comando compila o algoritmo e salva o arquivo binário resultante como "olamundo" (ou "olamundo.exe") no diretório atual. Se a opção "-o <arquivo>" não for informada, o GPT criará o executável usando o nome do algoritmo. Após executar o programa criado, é exibido o texto "Ola mundo!" na tela.

Como já deve ter assumido, comentários ao estilo C (/* */) e C++ (//) são permitidos e o que estiver entre aspas duplas (") é tratado como uma string ou constante literal. Escapes são permitidos como EOL (\n), tabulação (\t) dentre outros. Não é permitido a concatenação em múltiplas linhas como na linguagem C:


```
imprima("Isso é uma "  
        "Concatenação"); //erro!
```

Outro detalhe é a pontuação. Da mesma forma que em C, o “;” é utilizado como finalizador de enunciados. Se for necessário utilizar uma aspa dupla em um literal, deve-se utilizar o escape:

```
imprima("Isso é literal com aspas duplas ($\backslash$) ");
```

3.2 Variáveis

A declaração de variáveis deve ser feita dentro de um bloco específico, que deve aparecer logo após a declaração do algoritmo. O bloco é iniciado pela palavra-chave “variáveis” (sim, com acento) e termina com a palavra chave “fim-variáveis”. Pelo menos uma variável deve ser declarada dentro do bloco (embora o bloco em si seja opcional) e apenas um bloco em escopo global deve ser declarado. Eis um exemplo para estudo:

Programa 2 Declaração de variáveis globais.

```
algoritmo teste_variaveis;
```

```
variáveis  
  x      : inteiro;  
  nome   : literal;  
fim-variáveis
```

```
início  
fim
```

3.2.1 Variáveis primitivas

Variáveis primitivas são declaradas seguindo o seguinte modelo:

```
<identificador> [, identificador]* : <tipo>;
```

Isso é, um ou mais identificadores separados por vírgula, seguido de um “:”, seguido de um tipo, seguido, finalmente, de “;”. Como pode-se notar, é parecido com Pascal. Assim que um programa é executado, todas as variáveis, primitivas ou vetores/matrizes, declaradas são iniciadas com um valor nulo ou “0” automaticamente. Vale ressaltar que constantes (const, final, etc) não são suportados. Os tipos primitivos suportados encontram-se na tabela 2.1

A fim de explorar melhor os aspectos da declaração, seguem-se alguns comentários a respeito do último exemplo (programa 2)

- Observe que o nome do algoritmo (“teste_variaveis”) não tem acento. Se declarar o algoritmo como “teste_variáveis” e tentar compilar o código, o seguinte erro será exibido:

Linha: 1 - “teste_variáveis” não pode ter caracteres especiais.

Portanto, identificadores (nomes de variáveis, funções e do algoritmo) não podem ter acentos ou caracteres especiais como \$, #, etc. A definição de um identificador em G-Portugol é equivalente ao das linguagens populares: uma letra (a-z ou A-Z) seguido de qualquer número de letras ou números. Finalmente, underlines (_) são permitidos. Cedilhas, portanto, também não formam identificadores válidos.

- O segundo ponto, é a palavra-chave “variáveis”: ela tem acento, e isso é permitido e obrigatório.

- O terceiro, é a definição do bloco e sua (falta de) semelhança com o Pascal. Todas os blocos em G-Portugol tentam seguir o formato “nome/fim-nome”, em favor da uniformidade e em detrimento de exceções linguísticas que confundem os estudantes.
- E, finalmente, o quarto ponto é a ausência de código entre “início” e “fim”. O programa não é obrigado a ter enunciados ou comandos.

Para maiores detalhes, veja o capítulo 4, sobre a implementação da linguagem.

3.2.2 Vetores e matrizes (conjuntos)

Vetores e matrizes “n” dimensionais de tipos primitivos são suportados. Um exemplo de declaração de uma matriz:

```
variáveis
  bitset : matriz[10] de lógicos;
  quadr : matriz[4][4] de inteiros;
fim-variáveis
```

O tipo do vetor ou matriz é dado pelo nome do tipo no plural (“inteiros” para tipo inteiro, “literais” para tipo literal, etc). Os subscritos na declaração (delimitados por “[]”) indicam o tamanho da matriz, e sua dimensão é informada pelo número de subscritos. Portanto, “bitset” é um vetor de 10 valores lógicos, enquanto “quadr” é uma matriz bidimensional, onde as duas dimensões tem tamanho 4.

É importante observar que matrizes são “0 based”, isso é, iniciam no índice 0 e seus índices são sempre inteiros positivos. Logo, a matriz “bitset” pode ser usada do índice 0 até o índice 9 (inclusive). Seu índice 10 não é válido e seu uso poderá acarretar em erros de execução (*runtime errors*). Matrizes, assim como variáveis de tipos primitivos, são inicializadas com o valor “0” ou “nulo” em todas as suas posições. Quando usadas como argumentos de funções, matrizes são passadas *por valor*.

“As pessoas são divididas em dois grupos: aquelas que começam a contar a partir do '0', e aquelas que não.”

– Anônimo

3.3 Estruturas condicionais

Por enquanto, apenas a estrutura *se/então/senão* é suportada. Essa e as demais estruturas utilizam expressões, que são avaliadas para que uma decisão seja tomada (repetir execução, selecionar bloco de instruções, etc). Qualquer expressão pode ser avaliada como expressão lógica. Expressões numéricas de valor “0” são avaliadas como falso. Demais valores numéricos são avaliados como verdadeiro. Valores literais nulos, da mesma forma, são avaliados como falso, e demais textos, como verdadeiro. Para maiores detalhes sobre expressões, veja a seção 4.3.3.

```
enquanto x faça           //depende do valor de x
enquanto "nome" = "nome" faça //avalia como verdadeiro
enquanto verdadeiro faça  //avalia como verdadeiro
enquanto 3+5 faça         //avalia 8 como verdadeiro
enquanto "nome" faça      //avalia como verdadeiro
enquanto 0 faça           //avalia 0 como falso
enquanto "" faça          //avalia como falso
enquanto '' faça          //avalia como falso
```

O programa 3 ilustra um algoritmo simples que verifica se o usuário é maior de idade.

Alguns pontos a serem considerados:

Programa 3 Exemplo de programa que utiliza estrutura condicional.

```
algoritmo idade;

variáveis
    idade : inteiro;
    nome : literal;
fim-variáveis

início
    imprima("Digite seu nome:");
    nome := leia();

    imprima(nome, ", digite sua idade:");
    idade := leia();

    se idade >= 18 então
        se idade < 60 então
            imprima("adulto!");
        senão
            imprima("ancião", '!');
        fim-se
    senão
        imprima("menor", "!");
    fim-se
fim
```

-
- O nome do algoritmo é “idade”, assim como o nome de uma variável. Não há conflitos.
 - O operador de atribuição é o *pascal-like* “:=”.
 - A função interna “leia” pode ser usada por variáveis primitivas de qualquer tipo.
 - A função “imprima” recebe um número variável de argumentos de qualquer tipo primitivo, sendo que pelo menos um argumento deve ser passado, que podem ser constantes literais (números, textos entre aspas, caracteres entre aspas simples,...), variáveis primitivas ou índices de vetores/matrizes. Os valores são concatenados e adicionados de um caractere EOL (end of line, ou fim de linha).
 - Já vimos que strings/literais constantes são denotados por texto entre aspas duplas (“”). Tal qual em C, um caractere entre aspas simples (') é um caractere constante que, também, permite escapes para representar caracteres como EOL ('\n').
 - Assim como a palavra-chave “variáveis”, “então” e “senão” devem ser acentuadas.
 - Parêntesis ao redor da expressão da estrutura “se/então” são opcionais.
 - Como em “variáveis/fim-variáveis”, blocos “se” tem seus respectivos e obrigatórios “fim-se”. Não há a opção de ignorar a definição do bloco quando apenas um enunciado é usado, como na linguagem C onde o uso de “{ }” é opcional. Também, não há imposições sobre a indentação. O programa 4 ilustra esse assunto.

Programa 4 Uso incorreto de estrutura condicional.

```
//código inválido: faltando fim-se
algoritmo se_invalido;

início
    se x = 2 então
        imprima("ok");
    imprima("estou dentro ou fora do bloco \"se\"?");
fim
```

3.4 Estruturas de repetição

3.4.1 A estrutura “enquanto”

A estrutura “enquanto” é a mais básica e comum das estruturas de repetição. Seu funcionamento é o mesmo que em outras linguagens populares, onde um conjunto de instruções é executado repetidamente enquanto o valor lógico de uma dada expressão for avaliado como “verdadeiro”.

Programa 5 Exemplo de programa que utiliza a estrutura “enquanto”.

```
algoritmo fatorial;

variáveis
    res : inteiro;
    fat : inteiro;
    x : inteiro;
fim-variáveis

início
    imprima("Digite um número:");
    x := leia();

    fat := x;
    res := 1;

    enquanto x <> 0 faça
        res := res * x;
        x := x - 1;
    fim-enquanto

    imprima("fatorial de ",fat," é igual a ",res);
fim
```

Assim como na estrutura “se/então”, parênteses em volta da expressão são opcionais e as expressões seguem as mesmas regras.

3.4.2 A estrutura “repita”

A estrutura “repita” tem o funcionamento semelhante à “enquanto”. A diferença está na ordem da avaliação da condição de repetição: “enquanto” avalia a condição antes de iniciar o laço, e “repita” avalia ao final do bloco.

Programa 6 Exemplo de programa que utiliza a estrutura “repita.”

```
algoritmo fatorial2;

variáveis
    res : inteiro;
    fat : inteiro;
    x   : inteiro;
fim-variáveis

início
    imprima("Digite um número:");
    fat := leia();
    res := 1;
    x := fat;

    repita
        res := res * x;
        x   := x - 1;
    até x <= 0;

    imprima("fatorial de ",fat," é igual a ",res);
fim
```

3.4.3 A estrutura “para”

A estrutura “para” é uma especialização da estrutura “enquanto”, e costuma ser ensinada em cursos de programação. Sua sintaxe é semelhante ao que se vê em literatura sobre algoritmos e estruturas de dados, entretanto, é uma sintaxe diferente de linguagens populares como C e Java.

A estrutura “para” tem a seguinte forma:

```
para <variável> de <expressão> até <expressão> [passo <inteiro>] faça
    [lista de comandos]
fim-para
```

Onde:

- “variável” deve ser uma variável numérica;
- “expressão” deve ser uma expressão que tem seu valor avaliado como numérico;
- “passo”, se existir, deve ser seguido por um inteiro constante.

As expressões de/até controlam os valores que a variável numérica terá no início e no fim do laço, respectivamente. Tanto o controle da frequência, quanto a decisão de incrementar ou decrementar a variável de controle é feita pelo termo opcional “passo”, e seu valor padrão é 1. Por exemplo, para iterar o valor de uma variável numérica “x” de 0 até 10, escreve-se:

```
para x de 0 até 10 faça
    //comandos...
fim-para
```

Da mesma forma, para uma iteração decrescente, de 2 em 2, escreve-se:

```
para x de 10 até 0 passo -2 faça
    //comandos...
fim-para
```

Programa 7 Exemplo de programa que utiliza a estrutura “para”.

```
algoritmo fatorial;

variáveis
    res : inteiro;
    fat : inteiro;
    x   : inteiro;
fim-variáveis

início
    imprima("Digite um número:");
    fat := leia();

    res := 1;
    para x de fat até 1 passo -1 faça
        res := res * x;
    fim-para

    imprima("fatorial de ",fat," é igual a ",res);
fim
```

Em comparação com a estrutura “for” de linguagens com sintaxe baseadas em C, há diferenças não só de sintaxe, mas de implementação. Um “for” que itera sobre uma variável numérica de 0 até (incluindo) 10, ao sair do laço, o valor dessa variável será 11. Em G-Portugol, a variável terá o valor 10 ao sair do laço. Essa diferença acontece porque a sintaxe do “para” induz a esse comportamento, diferente da sintaxe do “for”, que tem um aspecto de mais baixo nível.

```
//código em C
for(x = 0; x <= 10; x++);
printf("%d", x); //imprime "11"
//-----

//código equivalente em G-Portugol
para x de 0 até 10 faça
    fim-para

imprima(x); //imprime "10"
```

Ademais, da mesma forma que o “for”, é possível que a variável de controle tenha seu valor alterado pelos comandos aninhados. Isso permite que o laço seja encerrado prematuramente, como também é comum em estruturas como “enquanto”. A utilidade dessa técnica está no fato de G-Portugol não incorporar mecanismos para refinar o controle de laços (como “break” e “continue”, encontrados em linguagens populares).

3.5 Funções

Subprogramas em G-Portugol são implementados no modelo de funções, que podem ter zero ou mais parâmetros de qualquer tipo, tanto primitivos quanto complexos (vetores e matrizes). Opcionalmente, elas podem definir valores de retorno, que deve ser de tipo primitivo. Tanto o retorno de dados como a passagem de argumentos são feitos *por valor*.

Para retorno prematuro ou retorno de dados, a palavra chave “retorne” é usada. Para funções que retornam dados, “retorne” deve ser seguido de um operando, que é uma expressão cujo valor deve ser compatível com o tipo da função. Já funções que não declaram um tipo de retorno explicitamente (equivalente a funções de retorno “void” em C), “retorne” deve ser usado sem operando.

Programa 8 Exemplo de algoritmo que utiliza funções.

```
algoritmo fatorial_recursivo;
```

```
variáveis
```

```
    x : inteiro;
```

```
fim-variáveis
```

```
início
```

```
    imprima("Digite um número:");
```

```
    x := leia();
```

```
    imprima("fatorial de ",x," é igual a ",fatorial(x));
```

```
fim
```

```
função fatorial(z:inteiro) : inteiro
```

```
início
```

```
    se z = 1 então
```

```
        retorne 1;
```

```
    senão
```

```
        retorne z * fatorial(z-1);
```

```
    fim-se
```

```
fim
```

3.5.1 Funções internas

Como já foi visto em exemplos anteriores, G-Portugol oferece duas funções internas: “leia” e “imprima”, que permitem uso básico de entrada e saída, respectivamente.

A função “leia” não recebe argumentos e retorna o valor lido da entrada padrão (“STDIN”), o que significa, geralmente, ler os dados que o usuário digitar no teclado, seguido do caractere “nova linha” (em geral, associado a tecla “Enter” no teclado). O tipo de dado retornado por “leia” é implicitamente convertido para o tipo primitivo exigido no contexto em que ela é usada.

A função “imprima” recebe um número variável de argumentos de qualquer tipo primitivo, sendo que pelo menos um argumento deve ser passado. Os valores passados como argumentos são convertidos para texto, concatenados na ordem definida e enviados para “STDOUT” (em geral, associado ao *display* ou monitor). Não há retorno de valor para esta função.

Programa 9 Exemplo de usos das funções internas “leia” e “imprima”.

```
algoritmo io;

variáveis
  c: caractere;
  i: inteiro;
  r: real;
  l: literal;
  z: lógico;
  mat: matriz[2] de inteiros;
fim-variáveis

início
  imprima("digite um caractere");
  c := leia();
  imprima("Digite um número inteiro");
  i := leia();
  imprima("Digite um número real:");
  r := leia();
  imprima("Digite um texto:");
  l := leia();
  imprima("Digite um valor lógico (\"verdadeiro\" ou \"falso\"), um número ou um texto:");
  z := leia();

  imprima("caractere: ",c," ", inteiro: ",i, ", real: ",r," ", texto: ",l, ", lógico: ", z,"\n");
fim
```

Capítulo 4

Implementação da linguagem G-Portugol

4.1 Introdução

Ao definir uma linguagem de programação voltada para o ensino de lógica e algoritmos, vários aspectos devem ser considerados. Ao contrário de linguagens de produção, não há preocupações como o poder expressivo da linguagem, dicionário em inglês, acesso a recursos de sistema, etc. A preocupação central está em oferecer uma ferramenta que:

- reflita os processos computacionais, exigindo o mínimo de conhecimento e experiência do estudante;
- evidencie os processos relacionados com o desenvolvimento de softwares;
- estimule a abstração e raciocínio lógico do estudante.

4.2 A linguagem

O ponto fundamental que guia as diretrizes da linguagem G-Portugol é seu propósito educacional: ela deve expressar processos computacionais de forma que um leigo os compreenda sem enfatizar a si mesma. Isso é, a linguagem em si deve chamar o mínimo de atenção possível, fazendo com que a compreensão dos processos computacionais seja tão natural quanto ler sua descrição informal, ou não-estruturada.

Esse objetivo encontra restrições, quando se leva em consideração a natureza das linguagens artificiais e o uso e forma popular atual da linguagem português, principalmente em literaturas. É de interesse que G-Portugol ofereça compatibilidade com essas formas, o que pode gerar conflitos quanto a decisões de design e restringir suas características. Portanto, embora as diretrizes marquem a base do design, muitas vezes, elas devem ser sacrificadas ou ajustadas para incorporar formas populares.

É interessante ressaltar que criar uma linguagem totalmente nova, que utiliza outros paradigmas como, por exemplo, orientação a objeto, possa ser interessantes e, talvez, mais eficientes como ferramentas de ensino, mas G-Portugol está, no momento, comprometida com a compatibilidade.

A seguir, alguns tópicos serão comentados quanto as diretrizes por trás das formas léxicas e gramaticais da linguagem G-Portugol.

4.2.1 Diretrizes para o design da linguagem

A linguagem deve ser totalmente em português e deve respeitar acentuações

Linguagens de programação, em geral, não se utilizam de caracteres especiais (ex. caracteres acentuados, cedilhas, e outros que não pertencem ao alfabeto inglês) para definição de seu dicionário, visto que são baseadas na língua inglesa. Portanto, letras acentuadas não são consideradas.

A decisão de incorporar palavras que respeitam a língua portuguesa é importante, visto que modificar a linguagem de forma a se afastar de sua língua natural (o português) pode evidenciar excessões as quais forçariam os usuários (estudantes e professores) a se ater mais com o estudo da linguagem do que com o estudo da disciplina em questão. Isso é, a ausência de acentos, por exemplo, obriga o usuário a aprender seus termos excepcionais. Além do mais, o uso de termos como “nao”, chama a atenção constante do usuário para o fato de a palavra não estar acentuada, o que costuma desviar atenção do estudo.

Consequentemente, o uso de acentos permite que a linguagem seja o mais próximo do português quanto for possível, apoiando a regra de não chamar atenção para si. Ademais, o código fica mais legível e permite uma leitura mais agradável.

Além dos acentos, é exigido que as palavras-chave usadas sejam completas ou por extenso, sem permitir abreviações (ex. “proc”, “func”, “char”, “int”, ...), o que dificulta a leitura de programas por um leigo.

Há também decisões quanto a forma verbal de comandos e funções. Em geral, na literatura, os verbos nos algoritmos são expressos no imperativo, visto que a linguagem é caracterizada como imperativa. Por outro lado, vale notar que, mesmo em linguagens imperativas, é comum ver programas que utilizam termos em português usando verbos no infinitivo.

Mesmo com essa perspectiva em vista, G-Portugol se utiliza de verbos no imperativo para seus termos, de forma a se aproximar das formas utilizadas nas literaturas sobre algoritmos.

“Programas devem ser escritos para pessoas lerem e, apenas incidentalmente, para máquinas executarem.”

– Abelson e Sussman

A linguagem deve ser simples, uniforme e coerente

Essa diretriz diz respeito a evitar excessões entre as formas gramaticais e comportamentos semânticos na medida do possível. Isso significa não só que as estruturas e comandos devem ser simples, claros e óbvios, como também, sua forma e comportamento devem ter um princípio em comum, evidenciando um formato uniforme. Naturalmente, a linguagem portugol já detém uma parcela relevante de características que se acomodam nessas diretrizes, no entanto, alguns detalhes ainda merecem atenção, como a representação de blocos estruturais.

A simplicidade implica em evitar construções e recursos que desviam a atenção do propósito original para detalhes da linguagem. Isso é, apenas recursos essenciais devem fazer parte de seu núcleo. Portanto, em um primeiro momento, não é interessante equipar a linguagem com recursos como alocação dinâmica de memória, controle de threads, etc.

Deve ser configurável naquilo que diz respeito a diferentes abordagens de ensino e estilo do professor

Essa diretriz diz respeito a permitir que a linguagem mude em certos aspectos (ou disponibilizar meios simples para realizar essas mudanças) de acordo com o gosto do professor, ou como modo de adaptar a forma de portugol de uma dada literatura. Dialeto e formas da linguagem portugol variam de literatura para literatura, e pode ser interessante permitir que a linguagem se adapte ou oferecer diferentes formas da linguagem.

Entre os aspectos que podem ser adaptados com facilidade se encontram os termos léxicos (palavras-chave, operadores, etc), e algumas formas de expressar estruturas, como algumas declarações e enuncia-dos.

Por exemplo, pode ser interessante que, para se declarar uma variável, não se use um bloco iniciado por “variáveis” e terminado por “fim-variáveis”, mas como um bloco parecido com a declaração de variáveis em Pascal, ou até em C (que sequer exige delimitação de bloco de variáveis).

Isso pode ser interessante quando se quer, por exemplo, ensinar uma linguagem específica como Pascal, fazendo com que G-Portugol possa usar estruturas e operadores semelhantes a esta linguagem, de forma a oferecer uma transição mais direta e que aproveita melhor os conhecimentos do estudante.

Atualmente, o programa GPT não implementa mecanismos para esse tipo de mudança, sendo necessário modificar seu código fonte e recompilá-lo.

4.3 Formato Estrutural

Um programa escrito em G-Portugol tem o seguinte formato:

- declaração do algoritmo
- declaração de variáveis globais
- bloco principal
- declaração de funções

A seguir, alguns pontos serão discutidos a respeito do formato da linguagem.

4.3.1 Declaração do algoritmo

A declaração do algoritmo não influencia o programa ou sua execução, visto que é apenas uma sessão informativa no código que, embora não seja um comentário, tem o mesmo efeito prático. A adoção dessa declaração pode ser discutida, e foi escolhida por ser bastante utilizada em literaturas.

4.3.2 Declaração de variáveis globais

Como visto no capítulo 2, essa declaração é opcional, e seu formato pode ser visto no exemplo a seguir:

```
variáveis  
  x : inteiro;  
fim-variáveis
```

Esse formato difere de linguagens como Pascal e C. Em Pascal, o bloco não tem um delimitador final (como “fim-variáveis”) e em C, não existe qualquer delimitador.

Delimitar o bloco permite maior consistência com outras formas gramaticais como os blocos de comando e estruturas de controle, e torna o código mais claro e explícito, embora adicione construções redundantes.

Os tipos primitivos englobam os tipos mais básicos das linguagens populares. No momento, agregados heterogêneos como, por exemplo, estruturas/registros (“struct” em C) não são suportados. Para uma lista dos tipos suportados, veja a tabela [2.1](#).

4.3.3 Bloco Principal

As linguagens de programação devem, de alguma forma, oferecer um “entry point” (ponto de entrada), de onde se inicia a execução do programa. O ponto de entrada pode ser uma função ou um bloco anônimo. Na literatura, em geral, o bloco principal é delimitado pelos termos “início” e “fim” e G-Portugol segue essa convenção. Essa decisão mantém um nível satisfatório de coerência com o bloco de variáveis globais e estruturas de controle, embora não sejam intimamente relacionados.

Vale ressaltar que não se faz imposição sobre a indentação do código. Esse recurso pode ser vantajoso no ensino, promovendo a clareza de código, portanto, sua implementação pode ser discutida para versões futuras.

Embora a sintaxe seja bem diferente, o bloco principal pode ser visto como uma função (como *main* na linguagem C). Há, portanto, a possibilidade de retornar um valor inteiro através do comando “retorne”.

Programa 10 Bloco principal com comando retorne.

```
algoritmo bloco_principal;

início
    imprima("Retornando valor do bloco principal");
    retorne 42;
    imprima("Essa linha não será impressa");
fim
```

Estruturas de Controle

Estruturas de controle são formadas por um cabeçalho seguido por um bloco de comandos e terminados por um delimitador final. Embora os delimitadores do bloco não sejam “início” e “fim”, há um grau de semelhança mantido: o delimitador inicial é omitido (uma vez que o cabeçalho é entendido como delimitador inicial) e o delimitador final é o termo “fim-” seguido do nome da estrutura.

No cabeçalho das estruturas como “enquanto” e “se”, as expressões não precisam ser delimitadas por parênteses.

Expressões

Expressões são operações que sintetizam, ao final, um valor. Em geral constituem valores ou operações aritméticas com um ou mais termos que podem ser variáveis, constantes ou chamadas a funções e são usadas em atribuições, estruturas de controle e subscritos de matrizes e vetores. Enunciados (como atribuições) não podem ser avaliados como expressões.

Constantes são valores *inline*, e cada tipo de dado tem uma forma de ser representada.

- **Inteiros:** Podem ser representados em base decimal, hexadecimal, octal e binária. Representações decimais são formadas, opcionalmente, por um sinal (“+” ou “-”) seguido de um mais algarismos(ex.“120”, “+5”, e “-2”). Representações hexadecimais são representadas com o prefixo “0x” ou “0X”, seguido de algarismos entre “0” e “9” e letras entre “a” e “f” ou “A” e “F” (ex. “0xF1A5”). Representações octais são representadas com o prefixo “0c” ou “0C”, seguido de algarismos entre “0” e “7” (ex. “0c61”). Finalmente, representações binárias são formadas pelo prefixo “0b” ou “0B”, seguido de algarismos “0” e “1” (ex. “0b101”).
- **Reais:** são representados por, opcionalmente, um sinal (“+” ou “-”), seguido de algarismos separados por um “.” como “-1.2345”.
- **Caracteres:** são representados por um único símbolo entre aspas simples. Alguns caracteres especiais são representados com escape (“\”) seguido de símbolo identificador. Esses caracteres são o LF (“\n”), CR (“\r”) e barra invertida (“\\”). A ausência de símbolos entre as aspas simples indica um caractere “nulo”. Internamente, caracteres são representados como números inteiros, o que permite sua compatibilidade numérica.
- **Literais:** são representados por um conjunto de caracteres entre aspas duplas. Eles podem conter caracteres especiais como “\n” e devem ser definidos em apenas uma linha de código. Valores literais são os únicos que não tem uma representação numérica, impedindo sua participação em expressões com operadores aritméticos (soma, divisão, etc). Comparações de igualdade para valores literais são feitas caractere por caractere em *case sensitive*. Portanto, a expressão “portugol” = “Portugol” é avaliada como falsa. Já comparações de grandeza são feitos calculando o número de caracteres que compõem os valores literais. Então, a expressão “maria” > “jósé” é avaliada como verdadeira.
- **Lógicos:** são representados pelas palavras “verdadeiro” e “falso”. Numericamente, qualquer valor diferente de “0” representa o valor verdadeiro e “0” representa o valor falso.

A precedência de operadores é mostrada na tabela 4.1 (da menor precedência para a maior) e pode ser explicitamente modificada com o uso de parênteses.

Precedência de Operadores	
Operador	Nome
ou	OR lógico
e	AND lógico
	OR binário
^	XOR binário
&	AND binário
=, <>	operadores igual e diferente
>, >=, <, <=	operadores relacionais maior, maior ou igual, menor, menor ou igual
+, -	operadores aritméticos soma e subtração
*, /, %	operadores aritméticos multiplicação, divisão, módulo
+, -, ~, não, ()	operadores unários positivo, negativo, NOT binário, NOT lógico, parênteses

Tabela 4.1: Precedência de Operadores

Em G-Portugol, não há coerção ou *casting* de tipos explicitamente. Todos os tipos numéricos (inteiro, real, lógico e caractere) são compatíveis entre si. É importante ressaltar que expressões envolvendo igualdade ou diferença de valores reais não são apropriadas dado a forma como esses valores são representados internamente. Tais comparações podem ter resultados imprevisíveis, e até diferentes, ao serem executados nos modos compilado, interpretado ou traduzido.

Existem dois casos em que ocorre uma coerção implícita. O primeiro caso ocorre durante a avaliação de uma expressão que tem operandos de tipos diferentes (mas compatíveis entre si), onde não há perda de dados e um dos termos é promovido para o tipo do seu termo complementar. Por exemplo, na expressão “2 + 1.5”, o termo “2” é promovido para o tipo “real”, tendo o valor “2.0” antes que a soma seja processada. Da mesma forma, sendo “x” uma variável de tipo real, a expressão “x := 5 / 2” atribuirá a “x” o valor “2.00”. Esse comportamento não muito óbvio é similar ao da linguagem C, onde “5 / 2” é avaliado como uma divisão de inteiros, onde apenas o valor inteiro final é relevante. Para obter o resultado real, a expressão deve ser “5.0 / 2”, para informar ao compilador que a divisão usará valores reais, produzindo a promoção de tipos mencionada anteriormente.

O segundo caso em que ocorre uma coerção implícita é discutido na seção 4.3.4.

4.3.4 Atribuições

Atribuições permitem a manipulação de valores na memória. Em G-Portugol, como já foi visto, é usado o operador “:=”, onde o termo a esquerda ou “*lvalue*” deve ser uma variável primitiva ou índice de uma matriz/vetor, e o termo a direita, uma expressão que, quando avaliada, tem seu tipo compatível com o *lvalue*.

Pode haver coerção de tipos durante a atribuição, quando o resultado da expressão é de um tipo diferente (mas compatível) do tipo de *lvalue*. É possível que dados sejam comprometidos, por exemplo, tendo uma expressão avaliada como real sendo atribuída a uma variável de tipo inteiro (o valor será truncado).

Matrizes não são aceitas como *lvalue*, como pode ser visto no programa 11.

4.4 Funções

Funções são os subprogramas de G-Portugol. São definidas após o bloco principal e podem receber argumentos e retornar valores. Em tempo de execução, as funções criam um novo escopo sobreposto ao escopo principal. Isso permite recursos como recursão e possibilita que variáveis locais (no escopo da função) tenham o mesmo nome que variáveis globais, onde essas últimas ficam “escondidas”, isso é,

Programa 11 Uso incorreto de matrizes.

```
algoritmo atribuicao_de_matrizes;
```

```
variáveis
  m1 : matriz[2] de inteiros;
  m2 : matriz[2] de inteiros;
fim-variáveis
```

```
início
  m1 := m2;    //erro
  imprima(m1); //erro
fim
```

incapazes de serem acessadas enquanto o escopo durar. Ressalta-se que não há suporte para funções aninhadas, isso é, funções declaradas dentro de funções.

O retorno de dados é feito por meio da instrução “retorne” e o valor de retorno (se houver) deve ser compatível com o tipo da função. Esse tipo não pode ser um tipo agregado como matrizes e vetores.

Tanto a passagem de argumentos quanto o retorno é feito *por valor*, isso é, a cópia do valor é feita, ao invés de a cópia de endereço ou passagem *por referência*.

As variáveis locais de uma função são formadas por seus parâmetros e pelas variáveis declaradas em seu escopo. A declaração de variáveis locais é feita entre o cabeçalho da função e a palavra-chave “início”, portanto, não é uniforme em relação a declaração global, onde se usa as palavras-chave “variáveis” e “fim-variáveis”.

A declaração dos parâmetros da função também não segue estritamente o formato de declaração de variáveis. O programa 12 ilustra a declaração de uma função.

Programa 12 Exemplo de usos de funções.

```
algoritmo exemplo_funcao;
```

```
início
  imprima(soma(2,2));
fim
```

```
função soma(x: inteiro, y: inteiro) : inteiro
  res : inteiro;
início
  res := x + y;
  retorne res;
fim
```

4.5 Funções internas

G-Portugol oferece duas funções internas para manipulação básica de entrada e saída. Ambas as funções tem comportamentos excepcionais quando comparadas com as funções de usuário. Essas funções são discutidas na sessão a seguir.

4.5.1 A função “imprima”

A função `imprima` tem como objetivo imprimir texto no dispositivo de saída. Ela recebe um número variável de argumentos (pelo menos um), onde cada argumento deve ser um valor primitivo, e os imprime em sequência. Ao final, é impresso um caractere de nova linha (LF) e a função retorna.

Essa função se comporta de forma excepcional visto que não é possível declarar funções em G-Portugol que recebem um número variável de argumentos.

4.5.2 A função “leia”

A função `leia` espera por uma entrada do teclado seguida do caractere LF (em geral, associado a tecla “Enter”) e retorna o valor lido. Esse valor é convertido implicitamente para o tipo do *lvalue*. A tabela 4.2 apresenta conversões implícitas processadas pela função `leia` para o enunciado de exemplo `x := leia()`.

Nota: números reais são arredondados, se necessário, e exibidos no formato .xx (com duas casas decimais). Portanto, o número “250.0” ou “250” seria impresso como “250.00” e “1.449” seria impresso como “1.45”. Se o parâmetro for uma variável, seu valor não será modificado.

Conversões da função “leia”		
Tipo do LValue “x”	Texto lido	Valor final de “x”
inteiro	123	123
inteiro	123 456	123
inteiro	abc	0
inteiro	123s	123
lógico	falso	falso
lógico	“0”	falso
lógico	falso 12wtc	verdadeiro
lógico	0 umdois3	verdadeiro

Tabela 4.2: Conversões implícitas da função “leia”

Essa função se comporta de forma excepcional visto que seu tipo de retorno não é absoluto (*overloaded*, depende do tipo de *lvalue*) e, no momento, é restrita expressões isentas de operadores. Isso é, não é permitido aplicá-la como um termo em uma expressão com múltiplos operandos (ex. `x := y + leia()`), embora seja possível utilizá-la em subscritos de vetores e matrizes, assim como em expressões de estruturas de controle.

Em princípio, pode-se levar em conta duas formas de implementar uma função de leitura em alto nível. A primeira, e mais simples é o uso de funções sobrecarregadas, que não retornam valor e recebem um parâmetro de um dado tipo (inteiro, real, etc), que é alimentado com o valor lido. Essa forma exige que a passagem seja feita por referência. A outra forma, é a implementada em G-Portugol, onde a função não recebe parâmetros e retorna o valor lido.

As duas formas exigem comportamentos excepcionais, visto que G-Portugol não tem suporte para sobrecarga de funções (por parâmetro ou retorno) ou passagem de parâmetros por referência.

Capítulo 5

O programa GPT

5.1 Introdução

GPT é a ferramenta principal da linguagem G-Portugol. Entre suas funções principais estão:

- Compilar algoritmos;
- Traduzir algoritmos para outras linguagens;
- Executar algoritmos.

Na versão atual é possível compilar algoritmos para sistemas Windows e Unices que suportam o formato ELF e o assembler NASM. Também, há suporte apenas para a tradução de algoritmos para a linguagem C. O resultado é um código em C 100% equivalente ao algoritmo.

É possível, também, executar algoritmos sem gerar código ou arquivos extra, de forma interpretada. Também, com a interpretação, é possível depurar passo a passo o algoritmo com o uso de uma ferramenta auxiliar (um cliente de depuração).

Vale ressaltar que, qualquer que seja o modo de uso do algoritmo (compilado, interpretado ou traduzido), se espera que eles tenham sempre comportamentos equivalentes. Isso é, o resultado de uma execução interpretada deve ser o mesmo se a execução fosse feita por meio de um binário compilado.

5.2 Opções gerais

Ao executar o programa “gpt” com o argumento “-h”, é mostrado:

Modo de uso: gpt [opções] arquivos

Opções:

- | | |
|--------------|---|
| -v | mostra versão do programa |
| -h | mostra esse texto |
| -o <arquivo> | compila e salva executável como <arquivo> |
| -t <arquivo> | salva o código em linguagem C como <arquivo> |
| -s <arquivo> | salva o código em linguagem assembly como <arquivo> |
| -i | interpreta |
| -d | exibe dicas no relatório de erros |

Maiores informações no manual.

As opções são comentadas a seguir.

- v: Exibe a versão do programa “gpt”, assim como informações de copyright;

- h: exibe todas as opções suportadas pelo programa “gpt”;
- o <arquivo>: Ao compilar um algoritmo, salva o executável com o nome de <arquivo>. Se essa opção for omitida na compilação, o nome do algoritmo será usado para criar o arquivo executável.
- t <arquivo>: Traduz o algoritmo para C, salva o código fonte com o nome de <arquivo>;
- s <arquivo>: Compila o algoritmo mas não cria código executável. Salva o código em assembly com o nome de <arquivo>;
- i: Executa o algoritmo diretamente, sem compilar ou criar arquivos. Opção conhecida como “interpretação” ou “scripting”.
- d: Exibe mais informações no relatório de erros, como dicas de como proceder para solucionar erros de sintaxe.

A última opção (“arquivos”) é uma lista de arquivos contendo o código fonte em G-Portugol, embora seja mais comum utilizar um arquivo apenas para cada algoritmo.

5.3 Tratamento de erros

Em geral, as ferramentas de diversas linguagens de programação oferecem o mínimo de informações a cerca de erros de compilação, às vezes, tendo uma forma “criptica” dificultando seu entendimento e posterior correção. Comum, também a possibilidade de que determinados erros sejam reportados em localizações distantes de onde o erro efetivamente se encontra no código fonte. Ademais, é importante notar que nessas linguagens, os erros são reportados em inglês.

O tratamento de erros é um aspecto importante do programa GPT. A reportagem de erros deve ser o mais claro possível e ter um formato uniforme, informando o local exato onde o erro foi localizado. Deve, também, fornecer dicas de como proceder para corrigir o erro, em alguns casos.

No momento, o analisador do GPT percorre o código fonte a procura de erros e, mesmo que encontre, continua a análise a procura de mais erros. Uma outra abordagem, seria interromper a análise assim que um erro fosse encontrado.

5.4 Execução de programas

5.4.1 Compilação e geração de código executável

O programa GPT é capaz de gerar código executável para arquiteturas compatíveis com x86 e em dois formatos: ELF (*Executable and linking format*) e PE (*Portable Executable*). Sistemas Unix, em geral, suportam o formato ELF, e o formato PE é conhecido nos sistemas Microsoft Windows. Após o processo de análise, o compilador gera código em Assembly para, então, usar o NASM (Netwide Assembler) como *backend* para montar e criar um executável válido. Consequentemente, não existe etapa de linkagem. A fase de otimização de código também não foi implementada.

Para usar esse recurso, é necessário que o NASM esteja instalado no sistema. Ele pode ser encontrado em <http://www.sf.net/projects/nasm>.

5.4.2 Tradução para a linguagem C

É possível usar o GPT para traduzir algoritmos escritos em G-Portugol para linguagem C. O código resultante, entretanto, não é criado com o intuito de ser facilmente lido, visto que a tradução não é direta. Isso é, o comportamento esperado pelo algoritmo deve refletir o comportamento que o código em C. Sendo assim, códigos extras são adicionados no arquivo resultante, para, por exemplo, permitir passagem de matrizes por valor, inicialização automática de variáveis e outras abstrações.

A tradução para C é limitada no que diz respeito a nomes e identificadores. É possível, por exemplo, declarar uma variável em G-Portugol com o nome de *printf*. Ao traduzir para C e tentar compilar o código resultante, o compilador pode emitir avisos e erros, visto que *printf* é uma função da biblioteca padrão, usada no código C resultante. Da mesma forma, identificadores com *underlines* antes ou depois em seus nomes (como “__leia_texto”) devem ser evitados, pois muitos identificadores internos utilizam essas convenções de nomeação, e seu uso pode acarretar em conflitos durante a tradução.

5.4.3 Interpretação de código

O programa GPT permite que o algoritmo seja executado sem gerar código binário. Esse modo é conhecido como “interpretação” e linguagens como Perl, PHP e Ruby utilizam esta técnica.

Esse modo permite depurar algoritmos passo a passo (por meio de um *client debugger* como o GPTEditor) e inspecionar variáveis e a pilha de funções enquanto o algoritmo está em execução.

A única diferença na execução de algoritmos em modo interpretado em relação a outros modos é que as matrizes/vetores tem seus subscritos checados (“*bound checking*”). Isso é, erros de execução são emitidos se um índice não existir em uma matriz/vetor.

Depuração interativa

A depuração interativa é feita em modo “interpretação”, portanto não gera código binário, executando o algoritmo diretamente. Para depurar interativamente um algoritmo é necessário um programa extra: o *client debugger*. Atualmente, o programa GPTEditor suporta a depuração interativa.

Entre os recursos disponíveis, pode-se citar a execução passo a passo em 3 modos (comumente conhecidas como “step into”, “step over” e “step out”), inspeção de variáveis locais/globais e pontos de parada (“*breakpoints*”).

A depuração ocorre tendo o programa GPT se comunicando via socket com o cliente (ex: GPTEditor), iniciando transmissão de dados entre esses dois pontos. As informações enviadas pelo GPT (variáveis, breakpoints, etc) usam o formato baseado em XML, enquanto o cliente envia comandos simples (o que evita que o programa GPT necessite de um XML parser).

Por exemplo, a pilha de funções pode ser representada da seguinte forma:

```
<stackinfo>
  <entry id="0"  function="@global" line="10"/>
  <entry id="1"  function="funcTeste" line="18"/>
</stackinfo>
```

5.4.4 Processando algoritmos divididos em múltiplos arquivos

A partir da versão 1.0, o GPT suporta processar algoritmos divididos em múltiplos arquivos. Esse recurso é possível utilizando duas formas:

- Passando os arquivos como opções na linha de comando;
- Utilizando a variável de ambiente GPT_INCLUDE.

A primeira forma é explicada na sessão 5.2. A segunda forma pretende facilitar a utilização de funções que devem estar disponíveis por padrão a cada execução/compilação dos algoritmos. Pode-se definir a variável de ambiente GPT_INCLUDE contendo vários caminhos de arquivos separados por “:”. Em sistemas Unix, por exemplo, pode-se criar a essa variável da seguinte forma (utilizando Bash):

```
$ export GPT_INCLUDE="/usr/local/lib/gpt/base.gpt:/usr/local/lib/gpt/util.gpt"
```

Onde “base.gpt” e “util.gpt” são arquivos contendo funções escritas em G-Portugol.

Os arquivos passados pela linha de comando e/ou que se encontram na variável GPT_INCLUDE são concatenados e processados como se o algoritmo estivesse em apenas um arquivo. Portanto, arquivos extras não devem ter declaração de algoritmo, bloco de variáveis globais ou bloco principal.

Apêndice A

Gramática da linguagem G-Portugol

A.1 Termos léxicos

A seguir é apresentado as convenções léxicas usadas em G-Portugol.

Regras para identificar literais numéricos

`T_INT_LIT : T_OCTAL_LIT | T_HEX_LIT | T_BIN_LIT | T_DEC_LIT`

`T_DEC_LIT : [0-9]+`

`T_OCTAL_LIT : '0' ('c'|'C') [0-8]+`

`T_HEX_LIT : '0' ('x'|'X') [0-9a-fA-F]+`

`T_BIN_LIT : '0' ('b'|'B') [01]+`

`T_REAL_LIT : T_DEC_LIT+ '.' T_DEC_LIT+`

Regras para identificar caracteres e cadeias de caracteres

`T_CARAC_LIT : ''' (~(''' | '\ ') | '\ ' .)? '''`

`T_STRING_LIT : '"' (~('"' | '\ ' | CR | LF) | '\ ' .)* '"'`

Regras para identificar comentários

`SL_COMMENT : "//" [^\n]* ('\n')?`

`ML_COMMENT : "/*" (~('*') | '*' ~ '/')* "*/"`

Regra para identificar nomes de variáveis, funções, etc.

`T_IDENTIFICADOR : [a-zA-Z_] [a-zA-Z0-9_]*`

A tabela [A.1](#) contém as palavras-chave padrão da linguagem G-Portugol.

A.2 Gramática

A seguir é apresentado a gramática da linguagem G-Portugol.

Palavras-chave de G-Portugol				
fim-variáveis	algoritmo	variáveis	inteiro	real
caractere	literal	lógico	início	verdadeiro
falso	fim	ou	e	não
se	senão	então	fim-se	enquanto
faça	fim-enquanto	para	de	até
fim-para	matriz	inteiros	reais	caracteres
literais	lógicos	função	retorne	passo

Tabela A.1: Palavras-chave de G-Portugol

```

algoritmo
: declaracao_algoritmo (var_decl_block)? stm_block (func_decls)* EOF
;

declaracao_algoritmo
: "algoritmo" T_IDENTIFICADOR ";"
;

var_decl_block
: "variáveis" (var_decl ";")+ "fim-variáveis"
;

var_decl
: T_IDENTIFICADOR ("," T_IDENTIFICADOR)* ":" (tp_primitivo | tp_matriz)
;

tp_primitivo
: "inteiro"
| "real"
| "caractere"
| "literal"
| "lógico"
;

tp_matriz
: "matriz" ("[" T_INT_LIT "]" )+ "de" tp_prim_pl
;

tp_prim_pl
: "inteiros"
| "reais"
| "caracteres"
| "literais"
| "lógicos"
;

stm_block
: "início" (stm_list)* "fim"
;

stm_list
: stm_attr

```

```

    | fcall ";"
    | stm_ret
    | stm_se
    | stm_enquanto
    | stm_repita
    | stm_para
    ;
stm_ret
: "retorne" expr? ";"
;

lvalue
: T_IDENTIFICADOR ("[" expr "]" ) *
;

stm_attr
: lvalue "!=" expr ";"
;

stm_se
: "se" expr "então" stm_list ("senão" stm_list)? "fim-se"
;

stm_enquanto
: "enquanto" expr "faça" stm_list "fim-enquanto"
;

stm_repita
: "repita" stm_list "até" expr ";"
;

stm_para
: "para" lvalue "de" expr "até" expr passo? "faça" stm_list "fim-para"
;

passo
: "passo" ("+"|"-" )? T_INT_LIT
;

expr
: expr ("ou"|"||" ) expr
| expr ("e"|"&&" ) expr
| expr "|" expr
| expr "^" expr
| expr "&" expr
| expr ("="|"<>" ) expr
| expr (">"|">="|"<"|"<=" ) expr
| expr ("+" | "-" ) expr
| expr ("/"|"*"|"%" ) expr
| ("+"|"-"|"~"|"não" )? termo
;

termo
: fcall

```

```
| lvalue
| literal
| "(" expr ")"
;

fcall
: T_IDENTIFICADOR "(" fargs? ")"
;

fargs
: expr ("," expr)*
;

literal
: T_STRING_LIT
| T_INT_LIT
| T_REAL_LIT
| T_CARAC_LIT
| T_KW_VERDADEIRO
| T_KW_FALSO
;

func_decls
: "função" T_IDENTIFICADOR "(" fparams? ")" (":" tb_primitivo)?
  fvar_decl
  stm_block
;

fvar_decl
: (var_decl ";")*
;

fparams
: fparam ("," fparam)*
;

fparam
: T_IDENTIFICADOR ":" (tp_primitivo | tp_matriz)
;
```