

PortuCol: uma pseudolinguagem inspirada em C ANSI para o Ensino de Lógica de Programação e Algoritmos

Lucas Lemos Barbosa, Ricardo Terra

Departamento de Ciência da Computação,
Universidade Federal de Lavras (UFLA), Brasil

`lbarbosa@computacao.ufla.br, terra@dcc.ufla.br`

Resumo. *Linguagens de programação são intrinsecamente complexas, devido à variabilidade, generalidade e completude de suas construções. Isso justifica a adoção de pseudolinguagens de alto nível no ensino de Lógica de Programação e Algoritmos. Embora C ANSI e Java estejam entre as linguagens de programação mais utilizadas, Portugol – uma pseudolinguagem com instruções em português inspirada na linguagem obsoleta Pascal – é ainda largamente adotada por Instituições de Ensino Superior (IES). Diante desse cenário, este artigo propõe PortuCol, uma pseudolinguagem de programação com instruções em português, assim como Portugol, porém inspirada em C ANSI. Este artigo também descreve PortuCol2C, uma ferramenta que traduz código escrito em PortuCol para C ANSI e conduz estudos empíricos, que demonstram, estatisticamente, que (i) PortuCol é mais similar a C ANSI do que o Portugol e (ii) PortuCol é para C ANSI o que Portugol é para Pascal.*

Abstract. *Programming languages are intrinsically complex due to the variability, generality, and completeness of their constructions. This justifies the adoption of high-level pseudocode languages when teaching Introduction to Algorithms and Programming Logic. Although ANSI C and Java are among the most used programming languages, Portugol—a pseudocode language with instructions in Portuguese, inspired by the outdated Pascal language—is still widely adopted by Brazilian universities. In view of such circumstances, this paper proposes PortuCol, a pseudocode programming language with instructions in Portuguese, as Portugol, although inspired by ANSI C. It also describes PortuCol2C, a tool that translates PortuCol to ANSI C, and conducts empirical studies, which statistically demonstrate that (i) PortuCol is more similar to ANSI C than Portugol and (ii) PortuCol is for C ANSI what Portugol is for Pascal.*

1. Introdução

No início do ensino superior, é comum que estudantes da área de Ciência da Computação e afins tenham dificuldade no raciocínio lógico para criação de soluções, uma vez que, no ensino médio, esses estudantes estavam acostumados a apenas aplicar fórmulas previamente existentes para a solução de problemas. Um estudo realizado em [3] atesta essa dificuldade, reportando que, em média, mais de 50% dos alunos das disciplinas de introdução à programação são reprovados a cada semestre.

Em virtude de linguagens de programação serem intrinsecamente complexas – devido à variabilidade, generalidade e completude de suas construções – docentes de Instituições de Ensino Superior (IES), usualmente, adotam pseudolinguagens de alto nível nas disciplinas de introdução à Lógica de Programação e Algoritmos. Pseudolinguagens proporcionam ao aluno um maior espaço para o desenvolvimento de seu raciocínio lógico relacionado à resolução de problemas, sem preocupações relativas ao uso correto de uma linguagem de programação.

Portugol é, hoje no Brasil, a pseudolinguagem mais adotada por universidades com esse propósito [12, 14], a qual conta com diversas ferramentas de apoio ao ensino [5, 7, 12, 10, 6]. No entanto, Portugol é o produto de uma simbiose do português e as linguagens de programação ALGOL e Pascal [7], ambas obsoletas [13]. Pascal, por exemplo, sequer está classificada entre as 50 linguagens de programação mais utilizadas no desenvolvimento de sistemas de software [13]. Além disso, Pascal apresenta alguns problemas em sua construção inerentes à linguagem, e.g., declaração das variáveis do bloco a priori. Assim, devido à origem sintática de Portugol, estudantes *não* terão uma transição natural ao aprender linguagens de programação do estado-da-prática, como C ANSI e Java.

Como uma alternativa ao Portugol, este artigo propõe PortuCol, uma pseudolinguagem de programação com instruções em português, assim como Portugol, porém inspirada em C ANSI. A escolha de C ANSI se justifica por pelo menos dois motivos: (i) juntamente com a linguagem Java são responsáveis por 35% de todos os sistemas de software em desenvolvimento [13]; e (ii) a linguagem C apresenta aplicações na educação, e.g., a plataforma de prototipagem eletrônica Arduino apresenta uma linguagem baseada em C [11] e pode ser usada em contextos de ensino e aprendizagem com o uso da robótica [2]. São apresentados estudos empíricos que atestam que (i) PortuCol é 27% mais similar a C ANSI do que o Portugol, o que proporciona uma transição mais natural para o estudante ao aprender linguagens do estado-da-prática e (ii) PortuCol é para C ANSI o que Portugol é para Pascal, o que retrata PortuCol como uma adaptação às linguagens do estado-da-prática. Além disso, este artigo descreve PortuCol2C, uma ferramenta que traduz código PortuCol para C ANSI, o que possibilita aos estudantes se familiarizem com a linguagem C ANSI através da observação do seu código PortuCol traduzido.

Este artigo está organizado como descrito a seguir. A Seção 2 apresenta Portugol, a pseudolinguagem mais adotada em disciplinas de programação do Brasil. A Seção 3 descreve o PortuCol, sua sintaxe e a ferramenta PortuCol2C. A Seção 4 reporta os resultados da avaliação de similaridade textual entre algoritmos implementados em PortuCol, C ANSI, Portugol e Pascal. Por fim, a Seção 5 conclui e propõe trabalhos futuros.

2. Portugol

O Portugol é a pseudolinguagem mais adotada em classes introdutórias à programação em IES no Brasil [12, 14]. Mais importante, a maioria dos livros utilizados como referência básica nas disciplinas de introdução à programação em universidades brasileiras possui explicações usando alguma notação de Portugol [12, 14]. Conclui-se que, o objetivo de Portugol é facilitar o aprendizado de Lógica de Programação e Algoritmos para alunos brasileiros não habituados com programação.

Portugol é o produto de uma simbiose do português e as linguagens de programação ALGOL e Pascal [7], ambas obsoletas [13]. Além disso, a linguagem Pascal tem problemas inerentes à linguagem. Um desses problemas é a declaração de variáveis que acontece antes do bloco de código ser iniciado, o que faz com que essas variáveis interfiram no bloco inteiro, contrariamente a somente interferir a partir do momento que são necessárias. Seu propósito é ser uma pseudolinguagem simples e com instruções em português para favorecer estudantes não familiares com a língua inglesa. Ao utilizar Portugol, o aluno não precisa se preocupar com a sintaxe e comandos em inglês, que uma linguagem de programação mais complexa possui, o que o permite focar sua concentração no *como* resolver o problema proposto.

Apesar de trazer benefícios no aprendizado de Lógica de Programação e Algoritmos, Portugol não provê, por definição, uma forma de executar o pseudocódigo escrito para que alunos possam testar seus algoritmos, o que se torna indispensável à medida que os algoritmos se tornam mais complexos e passam a requerer códigos mais elaborados. Esse problema ocorre pois o Portugol tem diversas variações em sintaxe e em comandos, dependendo do contexto de utilização. Isso faz com que as ferramentas não sejam padronizadas, i.e., suportando sintaxes e instruções diferentes [5, 7, 12, 10, 6].

Dentre as ferramentas existentes que apoiam o ensino utilizando Portugol podemos destacar: CIFluxProg [5], que disponibiliza um ambiente visual com fluxogramas, além de um ambiente textual com uma variação de Portugol; WEB-UNERJOL [7], que provê um ambiente *web* que permite o desenvolvimento e execução de algoritmos em Portugol através de um navegador; Portugol Studio [12], que é um ambiente de desenvolvimento integrado (IDE) para uma variante do Portugol que utiliza o compilador Portugol Núcleo; Portugol IDE [10], que provê um ambiente que possibilita desenvolver algoritmos em Portugol através do uso de fluxogramas como uma linguagem gráfica, além do tradicional modo textual; e, por fim, VisuAlg[6], que adota uma variante de Portugol bem parecida com Pascal, porém sem pontos-e-vírgulas para separar comandos. Os Algoritmos 1 e 2 ilustram tal similaridade apresentando um mesmo algoritmo escrito em Portugol e em Pascal, respectivamente.

Algoritmo 1. Portugol

```

1 algoritmo "helloWorld"
2 var
3     nome : caractere
4     i : inteiro
5 inicio
6     escreva ("Digite seu nome: ")
7     leia (nome)
8     para i de 0 ate 2 passo 1 faca
9         escreva ("Ola ", nome, "!")
10    fimpara
11 fimalgoritmo

```

Algoritmo 2. Pascal

```

1 Program helloWorld;
2 var
3     nome : string;
4     i : integer;
5 Begin
6     write ('Digite seu nome: ');
7     readln (nome);
8     for i := 0 to 2 do
9         Begin
10            write ('Ola ', nome, '!');
11        End;
12 End.

```

3. PortuCol

Este artigo propõe PortuCol, uma pseudolinguagem de programação com instruções em português, assim como Portugol, porém inspirada em C ANSI. O objetivo é fornecer uma ferramenta mais adequada ao ensino de Lógica de Programação e Algoritmos por meio

de uma pseudolinguagem cuja sintaxe seja inspirada em uma linguagem de programação do estado-da-prática (i.e., amplamente adotada por fábricas de software).

A Figura 1 ilustra o contexto de utilização de PortuCol. Estudantes desenvolvem o código em PortuCol e, automaticamente, a ferramenta PortuCol2C traduz PortuCol para C ANSI e executa o código correspondente. Isso possibilita que estudantes (i) executem seu pseudocódigo; (ii) se familiarizem com a linguagem C ANSI através da observação do seu código PortuCol traduzido; e (iii) tenham uma transição mais natural ao aprender linguagens do estado-da-prática como C ANSI e Java.

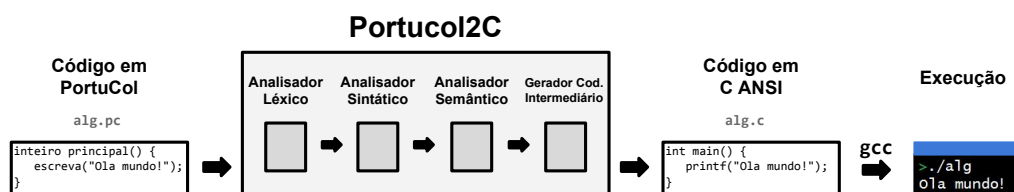


Figura 1. Utilizando PortuCol

3.1. Sintaxe

A definição da sintaxe de PortuCol considerou didática, facilidade de aprendizagem e similaridade com C ANSI, de forma que futuramente a transição para essa linguagem de programação seja mais natural para o aluno.

A Tabela 1 apresenta um subconjunto da equivalência de instruções entre PortuCol, C e Portugol. A tabela de equivalência de todas as instruções de PortuCol pode ser encontrada no Apêndice A. PortuCol possui instruções e palavras reservadas em português similarmente à Portugol e uma sintaxe similar à C ANSI. Em PortuCol, assim como em C ANSI, é necessário delimitar o escopo do código com os símbolos de abre e fecha chaves. Os operadores aritméticos e lógicos são como em C ANSI; exceções (e.g., operadores lógicos e e ou) ocorreram em virtude do caráter didático que a pseudolinguagem proposta tem como objetivo.

As estruturas básicas de desvio e repetição seguem, em sua maioria, a estrutura da linguagem de programação C ANSI. Contudo, uma das estruturas que foram modificadas com o intuito de apresentar um conceito de forma didática foi a estrutura de repetição para (for) que em PortuCol tem condições de parada fixas (menor ou igual, e maior ou igual). O objetivo dessa modificação foi de balancear a facilidade de aprendizado com o aspecto didático, presente também em Portugol, com a sintaxe de C ANSI que pode ser vista na estrutura do comando. Essa escolha também tem o objetivo de habituar o aluno a entender arranjos de tamanho n como estruturas que são acessadas a partir da posição 0 até a posição $n - 1$, o que acontece na maioria das linguagens de programação.

Usando o mesmo algoritmo descrito na Seção 2, os Algoritmos 3 e 4 apresentam dois exemplos do mesmo código em PortuCol e C ANSI, respectivamente.

Tabela 1. Equivalência de instruções PortuCol, C e Portugol

Construções	PortuCol	C	Portugol
Estrutura	inteiro principal(){ ... }	int main(){ ... }	Algoritmo "nome" var ... inicio fimalgoritmo
Variável tipo inteiro	inteiro i;	int i; short int i; long int i; long long int i;	i: inteiro
Ler valor	leia("%tipo", i);	scanf("%tipo", i);	leia(i)
Operador e	a e b	a && b	a e b
Condicional caso	escolha (variável) { caso valor1: ... caso valor2, valor3 ... padrao: ... }	switch (variável) { case valor1: ... break; case valor2: ... case valor3: ... break; default: ... break; }	escolha variável caso valor1 ... caso valor2, valor3 ... outrocaso ... fimescolha
Condicional para	para (variável = valor valorN ; P) { ... }	for (variável = valor variável <= valorN; variável += P) { ... }	para variável de valor1 ate valorN passo P faça ... fimpára

Algoritmo 3. PortuCol

```

1 inteiro principal(){
2     texto nome;
3     escreva("Digite seu nome: ");
4     leia("%texto", nome);
5     int i;
6     para( i=0 ; 2 ; 1){
7         escreva("Ola %texto!", nome);
8     }
9 }
```

Algoritmo 4. C ANSI

```

1 int main(){
2     char nome[20];
3     printf("Digite seu nome: ");
4     scanf("%s", nome);
5     int i;
6     for( i=0 ; i<=2 ; i++){
7         printf("Ola %s!", nome);
8     }
9 }
```

3.2. Ferramenta PortuCol2C

Para que os estudantes possam executar seus códigos escritos em PortuCol, foi criado um protótipo de uma ferramenta, denominada PortuCol2C. Essa ferramenta realiza a tradução dos códigos em PortuCol para a linguagem C ANSI e, por conseguinte, compila e gera o arquivo executável correspondente. Dessa forma, PortuCol2C permite que estudantes (i) “executem” seu código PortuCol e (ii) observem qual seria o código C ANSI correspondente, o que já os familiariza com linguagens de programação reais.

Na atual versão da ferramenta, a execução de PortuCol2C é por linha de comando, conforme pode ser observado na Figura 2. Dado um código em PortuCol (alg.pc), a execução ferramenta (portucol2c alg.pc) gera o código C ANSI correspondente (alg.c) e o arquivo executável (alg).

```

toy_example -- -bash -- 52x17
[toy_example@terra] $ cat alg.pc
inteiro principal(){
    escreva("Ola mundo!");
}

[toy_example@terra] $ ./portuCol2c alg.pc
Codigo C ANSI alg.c gerado e ja compilado com GCC.

[toy_example@terra] $ cat alg.c
#include <stdio.h>

int main () {
    printf("Ola mundo!");
}

[toy_example@terra] $ ./alg
Ola mundo!

```

Figura 2. Execução do PortuCol2C

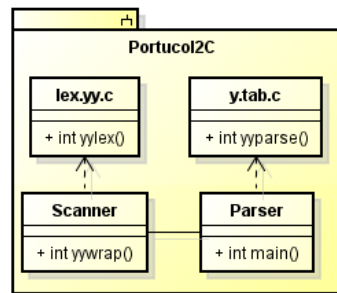


Figura 3. Arquitetura do PortuCol2C

O tradutor foi implementado usando *Lex* e *Yacc*,¹ que são ferramentas designadas para o desenvolvimento de compiladores e interpretadores [8]. O *Lex* realiza a análise léxica, que consiste na leitura dos caracteres de entrada e produção de uma sequência de *tokens*, enquanto o *Yacc* executa a análise sintática que consiste na obtenção de uma cadeia de *tokens* e verificação de quaisquer erros de sintaxe de uma forma inteligível [1]. Conforme ilustrado na Figura 3, a ferramenta segue então uma arquitetura com os seguintes dois módulos principais:

1. *Scanner*: Responsável por realizar a análise léxica do código-fonte escrito em PortuCol. Sua implementação consiste em definições de expressões regulares a serem convertidas em *tokens* tais como números, variáveis e palavras reservadas. Posteriormente, são definidas as regras para cada *token*, sendo que no caso do *Scanner* as regras estabelecidas consistem em retornar cada *token* para o *Parser*. A compilação do *Scanner* pelo *Lex* gera o arquivo `lex.yy.c`, o qual é utilizado para viabilizar a detecção dos *tokens* no código-fonte do PortuCol.
2. *Parser*: Responsável por realizar a análise sintática do código-fonte escrito em PortuCol. O *Parser* é constituído por regras formadas através dos *tokens* recebidos pelo *Scanner*, sendo que cada regra possui subrotinas associadas. No nosso caso, as subrotinas têm como finalidade imprimir, no arquivo de saída, a tradução para código C ANSI da sequência de *tokens* relacionado à cada regra. A compilação do *Parser* pelo *Yacc* gera o arquivo `y.tab.c`, o qual é utilizado para possibilitar a análise sintática dos *tokens*.

Os fontes dos analisadores léxico e sintático podem ser encontrados nos Apêndices B e C.

4. Avaliação

4.1. Questões de Pesquisa

Um estudo foi projetado para responder às seguintes questões de pesquisa:

QP #1 – PortuCol se aproxima à C ANSI mais do que Portugol?

QP #2 – PortuCol se aproxima à C ANSI mais do que Portugol se aproxima à Pascal?

¹<http://dinosaur.compilertools.net/>

O objetivo da QP #1 é avaliar se PortuCol se aproxima mais de linguagens do estado-da-prática, confirmando que o objetivo foi alcançado. Já a QP #2 tem como objetivo verificar se PortuCol é mais próxima às linguagens do estado-da-prática do que Portugol é à Pascal.

4.2. Dataset²

Para efetivamente comparar a similaridade entre algoritmos nas linguagens PortuCol, C, Portugol e Pascal, foram especificados 30 algoritmos, os quais foram implementados em cada uma das linguagens. A Tabela 3 descreve sucintamente os algoritmos implementados e aponta as principais estruturas utilizadas.

Tabela 3. Descrição dos algoritmos

Algoritmo	Descrição	Principais estruturas utilizadas
Alg_1	Cadastro e acesso de produtos	Procedimentos, enquanto, para, caso, se
Alg_2	Equação quadrática	Tipo de dado real, se senao se
Alg_3	Idade por ano de nascimento	Tipo de dado arranjo, para
Alg_4	Bubble sort	Tipo de dado arranjo, para, se
Alg_5	Busca binária	Função, enquanto, se senao se
Alg_6	Busca em arranjo	Faca enquanto, se, senao
Alg_7	Calculadora (+, -, *)	Caso, tipo de dado texto
Alg_8	Média de notas	Leitura de entrada, se, senao
Alg_9	Soma e subtração de reais	Para, se, senao
Alg_10	Combinação	Escrita e leitura de dados, Para
Alg_11	Decomposição de inteiro	Para, faca enquanto, se
Alg_12	Aprovação de aluno por nota	Caso
Alg_13	Fatorial recursivo	Função recursiva, se senao se
Alg_14	Fibonacci iterativo	Função, para
Alg_15	Selection sort	Para, tipo de dado arranjo, se
Alg_16	Mostrar coluna de matriz	Matriz 4x4, para
Alg_17	Média de reais	Função, tipo de dados real
Alg_18	Inverter texto	Tipo de dados texto, para
Alg_19	Números palíndromos	Enquanto, se, senao
Alg_20	Primos até certo número	Para, enquanto, se
Alg_21	Aproximação de Pi	Para, operação de raiz quadrada
Alg_22	Potenciação de inteiros (x^y)	Para
Alg_23	Comb sort	Enquanto, se, para
Alg_24	Soma de matrizes	Matrizes 2x2, para
Alg_25	Soma de algarismos	Enquanto, operação de resto de divisão
Alg_26	Classificação de texto por caso	Caso, comparação de texto
Alg_27	Validação de CPF	Para, se, conversão de texto para inteiro
Alg_28	Divisão de arranjos entre par e ímpar	Repita enquanto, para, se, senao
Alg_29	Aproximação de raiz quadrada	Se, enquanto, senao
Alg_30	Concatenação e comparação de texto	Se, senao, operações de concatenação e comparação de texto

4.3. Similaridade Textual

Para obter a similaridade entre os algoritmos implementados em PortuCol, C, Portugol e Pascal foi utilizada a similaridade de cosseno (*cosine similarity*) – técnica mais adotada em recuperação de informação e mineração de texto para comparação de documentos [9] – a qual mensura a similaridade de dois documentos com base no ângulo entre seus vetores representativos.³

Portanto, a similaridade de cosseno entre dois documentos d_1 e d_2 é computada a partir dos seus vetores representativos $\vec{V}(d_1)$ e $\vec{V}(d_2)$ conforme Equação 1.

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| |\vec{V}(d_2)|} \quad (1)$$

²Dataset publicamente disponível em: http://www.dcc.ufla.br/~terra/papers/2016_wei_portucol

³A similaridade varia de 0 (nada similares) a 1 (totalmente similares).

onde $\vec{V}(d_i)$ denota o vetor derivado de um documento d_i , com um componente no vetor para cada termo do dicionário. Assim, cada termo do documento é armazenado uma única vez no vetor e é associado a um peso. É importante enfatizar que os elementos presentes nos dois documentos estarão nos dois vetores representativos, apresentando peso zero no vetor representativo do documento em que o termo não está presente. O peso dos termos normalmente leva em consideração o $tf-idf_{t,d}$, onde tf é a frequência dos termos (*Term Frequency*), dada pela quantidade de vezes em que o termo aparece no documento em questão; idf é a frequência inversa do documento (*Inverse Document Frequency*), dada por uma função inversa que tem como objetivo diminuir o peso de termos que ocorrem com muita frequência em todos os documentos; t representa os termos e d o documento.⁴ Enfim, a similaridade entre dois documentos consiste na razão entre (i) o produto escalar e (ii) o produto das normas euclidianas dos dois vetores representativos, que se trata do cosseno do ângulo entre os dois vetores.

4.4. Metodologia

Foram conduzidas as seguintes atividades para se responder às questões de pesquisa:

1. Definir uma particular versão de Portugol, devido à existência de variações do Portugol encontradas na literatura, onde diferentes ferramentas [5, 7, 12, 10, 6] possuem suas próprias variações de Portugol. Assim, foi escolhida a variante adotada pelo VisuAlg – um interpretador de Portugol desenvolvido por Souza [6] – pois é uma ferramenta estável e com o objetivo de facilitar o aprendizado de programação, características que foram verificadas em [4].
2. Construir um *dataset* contendo uma vasta gama de algoritmos implementados nas linguagens PortuCol, C, Portugol e Pascal (veja Seção 4.2).
3. Comparar a similaridade textual – utilizando *cosine similarity* descrito na Seção 4.3 – dos algoritmos implementados na linguagem PortuCol com C, Portugol com C e Portugol com Pascal. A decisão de utilizar um algoritmo de similaridade textual, contrariamente à um algoritmo de similaridade estrutural, se deve ao fato de que, como os algoritmos implementados nas diferentes linguagens tinham as mesmas estruturas nas mesmas ordens, as estruturas seriam idênticas na grande maioria dos casos. É importante mencionar que textos referentes à inclusão de bibliotecas (e.g. `#include <stdio.h>` e `uses crt`) foram retirados dos códigos C e Pascal.
4. Determinar estatisticamente a relação de similaridade de PortuCol com C e de Portugol com C, a fim de responder a QP #1.
5. Determinar estatisticamente a relação de similaridade de PortuCol com C e de Portugol com Pascal, a fim de responder a QP #2.

4.5. Resultados

A Tabela 4 reporta os resultados da similaridade textual dos algoritmos implementados na linguagem Portugol com C (coluna 1), PortuCol com C (coluna 2) e Portugol com Pascal (coluna 3).

⁴Particularmente neste artigo, foi considerada apenas a frequência dos termos, uma vez que diminuir o peso de termos frequentes não é desejável.

Tabela 4. Similaridade Textual

Algoritmo	Portugol/C	PortuCol/C	Portugol/Pascal
Alg_1	0.4004	0.6031	0.4365
Alg_2	0.7009	0.6824	0.7607
Alg_3	0.6348	0.7314	0.7291
Alg_4	0.7824	0.8713	0.7850
Alg_5	0.6803	0.7712	0.6591
Alg_6	0.5866	0.7072	0.6256
Alg_7	0.6674	0.6292	0.7227
Alg_8	0.6190	0.5930	0.6347
Alg_9	0.3036	0.7593	0.6543
Alg_10	0.7386	0.6616	0.8338
Alg_11	0.7262	0.8586	0.7523
Alg_12	0.2761	0.3639	0.5355
Alg_13	0.6385	0.6468	0.6156
Alg_14	0.6230	0.6863	0.5823
Alg_15	0.7190	0.8915	0.7118
Alg_16	0.5420	0.6057	0.5809
Alg_17	0.4449	0.4355	0.5536
Alg_18	0.6502	0.7662	0.7963
Alg_19	0.8511	0.8685	0.8348
Alg_20	0.3073	0.7611	0.7110
Alg_21	0.3267	0.7708	0.7780
Alg_22	0.6261	0.6199	0.6581
Alg_23	0.8580	0.9192	0.8247
Alg_24	0.0407	0.8187	0.9045
Alg_25	0.1079	0.8729	0.7621
Alg_26	0.6490	0.7286	0.7338
Alg_27	0.6202	0.8472	0.5964
Alg_28	0.5844	0.6647	0.6306
Alg_29	0.8454	0.8956	0.8604
Alg_30	0.4611	0.5484	0.4665
Média (\bar{x})	0.5671	0.7193	0.6910
Dev. Pad. (s)	0.2089	0.1361	0.1161

QP #1 – PortuCol se aproxima à C ANSI mais do que Portugol?

Foram comparadas as similaridades textuais dos algoritmos implementados (i) em Portugol e em C e (ii) em PortuCol e em C, cujos resultados podem ser observados nas primeiras duas colunas da Tabela 4 e nas duas primeiras linhas da Figura 4.

O *dataset* disponível para avaliação apresenta os mesmos algoritmos implementados em diferentes linguagens (veja Seção 4.2). Isso permite que seja conduzida uma observação pareada dos resultados de similaridade textual. Portanto, considerando uma distribuição normal dos resultados, a distribuição probabilística *t-Student* foi utilizada no cálculo do intervalo de confiança para responder à QP #1, conforme Equação 2.⁵

O *dataset* utilizado possui trinta algoritmos ($n = 30$). A diferença da similaridade textual entre cada algoritmo Portugol/C e PortuCol/C (colunas 1 e 2 da Tabela 4) foi calculada. Assim, obteve-se a média aritmética ($\bar{x} = 0,1523$) e o desvio padrão ($s = 0,2158$) das diferenças. A confiança adotada para as amostras da distribuição de *t-Student* foi 99,9% com grau de liberdade de 29, onde $t_{[0,9995;29]}$ é igual a 3,659.

$$IC = \bar{x} \pm t_{[1-\frac{\alpha}{2};n-1]} \left(\frac{s}{\sqrt{n}} \right) = 0,1523 \pm 3,6590 \left(\frac{0,2158}{\sqrt{30}} \right) = [0,0081, 0,2964] \quad (2)$$

Assim, o teste *t* resultou no intervalo de $[0,0081, 0,2964]$, o qual *não* inclui zero. Isso indica que, com 99,9% de confiança, PortuCol se aproxima mais à C ANSI do que Portugol.

QP #2 – PortuCol se aproxima à C ANSI mais do que Portugol se aproxima à Pascal?

Similarmente à QP #1, foram comparadas as similaridades textuais dos algoritmos

⁵O teste *t* resulta em um intervalo de confiança. Caso o intervalo inclua zero, as amostras *não* são estatisticamente diferentes. Caso contrário, pode-se afirmar que as amostras *são* estatisticamente diferentes, o que, neste caso, indica que a similaridade em um par de linguagens é maior que na outra.

implementados (i) em PortuCol e em C e (ii) em Portugol e em Pascal, cujos resultados podem ser observados nas últimas duas colunas da Tabela 4 e nas duas últimas linhas da Figura 4. É importante mencionar que os resultados comparativos entre PortuCol e C já tinham sido previamente calculados na resposta à QP #1.

Considerando uma distribuição normal dos resultados, a distribuição probabilística *t-Student* novamente foi utilizada no cálculo do intervalo de confiança para responder à QP #2, conforme Equação 3. Similarmente à QP #1, a confiança adotada para as amostras da distribuição de *t-Student* foi 99,9% com grau de liberdade de 29.

$$IC = \bar{x} \pm t_{[1-\frac{\alpha}{2}; n-1]} \left(\frac{s}{\sqrt{n}} \right) = 0,0283 \pm 3.659 \left(\frac{0.1006}{\sqrt{30}} \right) = [-0,0389, 0,0955] \quad (3)$$

Assim, o teste *t* resultou no intervalo de $[-0,0389, 0,0955]$. Como o intervalo de confiança inclui zero, podemos considerar que o PortuCol se aproxima à C ANSI da mesma maneira que Portugol se aproxima à Pascal. A Figura 4 representa visualmente o intervalo de confiança para a média das similaridades textuais de cada par de linguagens.

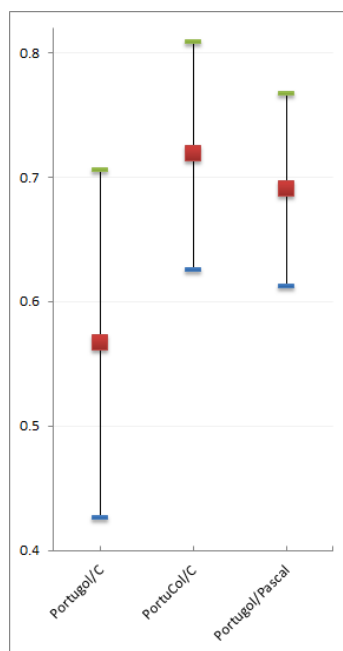


Figura 4. Intervalo de confiança

4.6. Ameaças à Validade

É importante mencionar pelo menos uma ameaça à validade de construção. A avaliação de similaridade entre PortuCol, C e Portugol analisou 30 algoritmos. A escolha desses algoritmos pode ter favorecido ou desfavorecido PortuCol na análise de similaridade. No entanto, foram escolhidos os 30 sistemas com as mais diversas estruturas presentes em sua construção e de forma completamente aleatória.

5. Considerações Finais

O uso de pseudolinguagens no ensino de Lógica de Programação e Algoritmos é uma estratégia muito utilizada em IES. Embora C ANSI e Java sejam as linguagens de programa-

ção mais utilizadas no desenvolvimento de sistemas de software [13], a pseudolinguagem mais utilizada no Brasil é o Portugol [12, 14], que é inspirada no obsoleto Pascal.

Diante desse cenário, este artigo propôs PortuCol, uma pseudolinguagem de programação com instruções em português assim como Portugol, porém inspirada em C ANSI. Como contribuição acadêmica, uma análise estatística da similaridade textual demonstrou que: (i) a pseudolinguagem PortuCol é mais similar à C ANSI do que Portugol, o que proporciona uma transição mais natural ao estudante ao aprender linguagens do estado-da-prática como C ANSI e Java; e (ii) PortuCol é para C ANSI o que Portugol é para Pascal, o que retrata apenas uma adaptação da pseudolinguagem às linguagens do estado-da-prática. Como contribuição prática, este artigo também apresenta PortuCol2C, uma ferramenta que executa código PortuCol através da prévia tradução para código C ANSI, o que possibilita aos estudantes se familiarizem com a linguagem C ANSI através da observação do seu código PortuCol traduzido.

O *dataset* dos algoritmos utilizados na avaliação e a ferramenta PortuCol2C (binário e código fonte) estão publicamente disponíveis em:

http://www.dcc.ufla.br/~terra/papers/2016_wei_portucol

Como trabalhos futuros, pretende-se (i) avaliar o uso da pseudolinguagem PortuCol no ensino de Lógica de Programação e Algoritmos; (ii) melhorar a ferramenta PortuCol2C com um ambiente de desenvolvimento integrado (IDE) e um depurador; e (iii) realizar a verificação de similaridade textual na formalização das gramáticas das linguagens, potencialmente no Formalismo de Backus-Naur (BNF).

Agradecimentos

Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

Referências

- [1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compiladores: Princípios, técnicas e ferramentas*. LTC, 2 edition, 2008.
- [2] Rafael Machado Alves, Armando Luiz Costa da Silva, Marcos de Castro Pinto, Fabio Ferrentini Sampaio, and Marcos da Fonseca Elia. Uso do hardware livre arduino em ambientes de ensino-aprendizagem. *Jornada de Atualização em Informática na Educação*, 1(1):162–187, 2013.
- [3] Ricardo Luiz B. L. Campos. Lógica de programação: Há como melhorar o aprendizado fugindo dos padrões estabelecidos nos livros didáticos e adotados pela maioria dos docentes? In *XVII Congresso Iberoamericano de Educación Superior em Computación (CIESC)*, pages 22–25, 2009.
- [4] Sormany Silva Dantas, Luiz Augusto de Macedo Morais, Rivanilson da Silva Rodrigues, Rodrigo Alves Costa, et al. Avaliação de um software educacional de apoio à aprendizagem de programação: Visualg. In *VII CONNEPI-Congresso Norte Nordeste de Pesquisa e Inovação*, 2012.
- [5] Rafael de Santiago and Rudimar Luís Scaranto Dazzi. Ferramenta de apoio ao ensino de algoritmos. In *XIII Seminário de Computação (SEMINCO)*, pages 1–8, 2004.
- [6] Cláudio Morgado de Souza. VisuAlg - ferramenta de apoio ao ensino de programação. *TECCEN*, 2(2):1–9, 2009.
- [7] Mauri Ferrandin and Simone Lilian Stephani. Ferramenta para o ensino de programação via internet. In *I Congresso Sul Catarinense de Computação (SULCOMP)*, pages 1–8, 2005.
- [8] John R. Levine, Tony Mason, and Doug Brown. *Lex & Yacc*. O’Reilly, 2 edition, 1992.

- [9] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.
- [10] António Manso, Luís Oliveira, and Célio Gonçalo Marques. Ambiente de aprendizagem de algoritmos – Portugol IDE. In *VI Conferência Internacional de TIC na Educação*, pages 969–983, 2009.
- [11] Michael McRoberts. Arduino básico. *Editora Novatec*, 344755160, 2011.
- [12] Luiz F. Noschang, Fillipi Pelz, Elieser A. de Jesus, and André L. A. Raabe. Portugol Studio: Uma IDE para iniciantes em programação. In *XXII Workshop sobre Educação em Computação (WEI)*, pages 1287–1296, 2014.
- [13] TIOBE. Programming languages index. http://www.tiobe.com/tiobe_index, March 2016.
- [14] Adriana Salvador Zanini and André Luís Alice Raabe. Análise dos enunciados utilizados nos problemas de programação introdutória em cursos de ciência da computação no Brasil. In *XX Workshop sobre Educação em Computação (WEI)*, 2012.

A. Equivalência de instruções entre PortuCol, C e Portugol

A Tabela 5 apresenta a equivalência de instruções entre PortuCol, C e Portugol.

B. Analisador Léxico

A Listagem 5 apresenta o código do analisador léxico, parte do módulo *Scanner*.

Listagem 5. Analisador léxico

```

1  %{
2  /*----- Definitions -----*/
3  #include <stdio.h>
4  #include "y.tab.h"
5
6
7  /*Prototipo*/
8  void installID(){
9
10 }
11 }
12 %}
13
14 abrechave      " ("
15 fechachave     ") "
16 abrecolchete  "[ "
17 fechacolchete "]" "
18 abreparentese "(" "
19 fechaparentese)" "
20 underline     "_ "
21 doispontos    ":" "
22 aspasimples   "\"" "
23 exclamacao    "!" "
24 string        "\"[^\\n]*[\\n]
25 caractere     '([^\\n|\\.|\\\\,)'
26 raizquadrada  "raizquadrada"
27 concatena     "concatena"
28 potencia      "potencia"
29 maiuscula     "maiuscula"
30 tamanhotexto "tamanhotexto"
31 e             "e"
32 ou            "ou"
33 somasub      ("+" | "-")
34 digito       [0-9]
35 numint       {digito}+
36 letra        [a-zA-Z]
37 se           "se"
38 senao        "senao"
39 faca         "faca"
40 enquanto     "enquanto"
41 para         "para"
42 retorne      "retorne"
43 inteiro      "inteiro"
44 real         "real"
45 texto        "texto"
46 logico       "logico"
47 verdadeiro   "verdadeiro"
48 falso        "falso"
49 semretorno   "semretorno"
50 escolha     "escolha"
51 caso        "caso"
52 padrao      "padrao"
53 interrompa  "interrompa"
54 principal   "principal"
55 escreva     "escreva"
56 leia        "leia"
57 porreal     "%real"
58 portexto    "%texto"
59 porinteiro  "%inteiro"
60 id          {letra}({letra}|{digito})|({letra}|{digito}|{underline})+
61 numreal     {digito}+\\.({digito}+)?([E][+-]?{digito}+)?
62 multdiv     ("*" | "/" | "%" )

```

Tabela 5. Equivalência de instruções PortuCol, C e Portugal

Construções	PortuCol	C	Portugal
Estrutura	inteiro principal(){ ... }	int main(){ ... }	Algoritmo "nome" var inicio fimalgoritmo
Variável tipo inteiro	inteiro i;	int i; short int i; long int i; long long int i;	i: inteiro
Variável tipo real	real d;	float d; double d; long double d;	d: real
Variável tipo lógico	logico b;	bool b; (C99)	b: logico
Variável tipo caractere	texto s;	char s;	s: caractere
Variável tipo <i>string</i>	texto s;	char s[];	s: caractere
Variável tipo vetor	tipo arranjo[linhas];	tipo arranjo[linhas];	v: arranjo[linha1..linhaN] de tipo
Variável tipo matriz	tipo arranjo[linhas][colunas];	tipo arranjo[linhas][colunas];	v: arranjo[linha1..linhaN, colunaa1..colunaaN] de tipo
Ler valor	leia("%tipo", i);	scanf("%tipo", i);	leia(i)
Escrever	escreva("...");	printf("...");	escreva("...")
Escrever com quebra de linha	escreva("...\n");	printf("...\n");	escreval("...")
Escrever com parâmetros	escreva("... %tipo ... %tipo ...", a, b);	printf("... %tipo ... %tipo ...", a, b);	escreval("...", a, "...", b, "...")
Operação de atribuição	a = b;	a = b;	a <- b a := b
Operação de adição	a + b;	a + b;	a + b
Operação de subtração	a - b;	a - b;	a - b
Operação de multiplicação	a * b;	a * b;	a * b
Operação de divisão	a / b;	a / b;	a / b
Operação de divisão inteira	a / b;	a / b;	a div b
Operação de resto da divisão	a % b;	a % b;	a % b a mod b
Operação de potenciação	potencia(a, b);	pow(a, b) <math.h>	a ^ b
Operação de raiz quadrada	raizquadrada(a);	sqrt(a); <math.h>	RaizQ(a)
Operação de concatenação	concatena("texto", "texto"); concatena("texto", a); concatena(a, "texto");	strcat("texto", "texto"); strcat("texto", a); strcat(a, "texto");	"texto" + "texto" "texto" + a (string) a (string) + "texto"
Operação de tamanho do texto	tamanhotexto(a);	strlen(a); <string.h>	compr(a)
Operador menor	a < b;	a < b;	a < b
Operador menor ou igual	a <= b;	a <= b;	a <= b
Operador maior	a > b;	a > b;	a > b
Operador maior ou igual	a >= b;	a >= b;	a >= b
Operador igual	a == b;	a == b;	a = b
Operador diferente	a != b;	a != b;	a <> b
Operador e	a e b	a && b	a e b
Operador ou	a ou b	a b	a ou b
Operador de negação	!a	!a	nao a
Condicional se	se (condição) { ... } senão { ... } fimse	if (condição) { ... } else { ... }	se condição entao ... senao ... fimse
Condicional aninhado	se (condição) { ... } senão se (condição) { ... } senão { ... } fimse	if (condição) { ... } else if (condição) { ... } else { ... }	se condição entao ... senao se condicao entao ... senao ... fimse
Condicional caso	escolha (variável) { caso valor1: ... caso valor2, valor3: ... padrao: ... }	switch (variável) { case valor1: ... case valor2: ... case valor3: ... default: ... }	escolha variável caso valor1 ... caso valor2, valor3 ... outrocaso ... fimescolha
Condicional enquanto	enquanto (condição) { ... }	while (condição) { ... }	enquanto condição faca ... Fimenquanto
Condicional para	para (variável = valor1 ; valorN ; P) { ... }	for (variável = valor1; variável <= valorN; variável += P) { ... }	para variável de valor1 ate valorN passo P faca ... fimpara
Comando de interrupção	interrompa;	break;	interrompa
Criar procedimento	semretorno nome(tipo parâmetro, ...) { ... }	void nome(tipo parâmetro, ...) { ... }	procedimento nome(parâmetro: tipo; ...) var ... inicio ... fimprocedimento
Criar função	tipo nome(tipo parâmetro, ...) { ... retorne valor; }	tipo nome(tipo parâmetro, ...) { ... return valor; }	funcao nome(parâmetro: tipo; ...): tipo var ... inicio ... retorne valor fimfuncao

```

63 |relacional          ("<"|">"|"=="|">="|"<="|"!=")
64 |atribuicao          "="
65 |comentario         "/.*"((\*+[/\*])|([^\*]))*\**"/"
66 |virgula            ","
67 |pontoevirgula     ";"
68 |delim              [ \t\n]
69 |ws                 {delim}+
70
71 |other              .
72
73 %%
74 %{
75 ----- Rules -----*/
76 %}
77 {ws}                { /*sem acao, sem retorno */ }
78 {comentario}        {
79                     int i;
80                     for(i=0;i<yy leng;i++){
81                         if(yytext[i]=='\n');
82                     }
83
84 {string}             { return STRING; }
85 {caractere}          { return CARACTERE; }
86 {abrechave}          { return ABRECHAVE; }
87 {fechachave}         { return FECHACHAVE; }
88 {abrecolchete}       { return ABRECOLCHETE; }
89 {fechacolchete}      { return FECHACOLCHETE; }
90 {abreparentese}      { return ABREPARENTESE; }
91 {fechaparentese}     { return FECHAPARENTESE; }
92 {doispontos}         { return DOISPONTOS; }
93 {aspasimples}        { return ASPASIMPLES; }
94 {exclamacao}         { return EXCLAMACAO; }
95 {raizquadrada}       { return RAIZQUADRADA; }
96 {concatena}          { return CONCATENA; }
97 {potencia}           { return POTENCIA; }
98 {maiuscula}          { return MAIUSCULA; }
99 {tamanhotexto}       { return TAMANHOTEXTO; }
100 {e}                  { return E; }
101 {ou}                 { return OU; }
102 {real}               { return REAL; }
103 {texto}              { return TEXTO; }
104 {inteiro}            { return INTEIRO; }
105 {logico}             { return LOGICO; }
106 {verdadeiro}         { return VERDADEIRO; }
107 {falso}              { return FALSO; }
108 {porinteiro}         { return PORINTEIRO; }
109 {porreal}            { return PORREAL; }
110 {portexto}           { return PORTEXTO; }
111 {semretorno}         { return SEMRETORNO; }
112 {principal}          { return PRINCIPAL; }
113 {escreva}            { return ESCREVA; }
114 {leia}               { return LEIA; }
115 {escolha}            { return ESCOLHA; }
116 {caso}               { return CASO; }
117 {padrao}             { return PADRAO; }
118 {interrompa}         { return INTERROMPA; }
119 {se}                 { return SE; }
120 {senao}              { return SENAO; }
121 {faca}               { return FACA; }
122 {enquanto}           { return ENQUANTO; }
123 {para}               { return PARA; }
124 {retorne}            { return RETORNE; }
125 {id}                 { installID(); }
126
127 {atribuicao}          { return ATRIBUICAO; }
128 {multdiv}            { return MULTDIV; }
129 {somasub}            { return SOMASUB; }
130 {numint}             { return NUMINT; }
131 {numreal}            { return NUMREAL; }
132 {virgula}            { return VIRGULA; }
133 {pontoevirgula}      { return PONTOEVIRGULA; }
134 {relacional}         { return RELACIONAL; }
135
136 {other}              { /*sem acao, sem retorno */ }
137 %%
138 ----- User subrotines -----*/
139 %}
140 int yywrap() {
141     return 1;
142 }

```

C. Analisador Sintático

A Listagem 6 apresenta o código do analisador sintático, parte do módulo *Parser*.

Listagem 6. Analisador sintático

```

1  %{
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <stdbool.h>
6  #include "replace.h"
7  #define YYSTYPE char*
8  int yylex(void);
9  void yyerror(char *);
10 extern FILE *yyin, *yyout;
11 extern char* yytext;
12
13 struct table {
14     char tipo[10];
15     char nome[30];
16     char vet[100];
17     int flag_param;
18     int flag_char;
19     int flag_string;
20 };
21
22 struct table tabela[300];
23 int current_tab_tipo = 0, current_tab_nome = 0;
24 int ident = 0;
25 %}

```

```

26 | %token ABRECHAVE FECHACHAVE ABRECOLCHETE FECHACOLCHETE ABREPRENTESE FECHAPRENTESE
27 | %token DOISPontos ASPASIMPLES EXCLAMACAO RAIZQUADRADA CONCATENA POTENCIA MAIUSCULA TAMANHOTEXTO E OU SOMASUB
   | NUMINT NUMREAL
28 | %token SE SENAO FACA ENQUANTO PARA RETORNE INTEIRO REAL TEXTO LOGICO VERDADEIRO FALSO SEMRETORNO ESCOLHA CASO
   | PADRAO INTERROMPA PRINCIPAL LEIA ESCREVA PORREAL PORTEXTO PORINTEIRO ID
29 | %token MULTDIV RELACIONAL ATRIBUICAO COMENTARIO VIRGULA PONTOEVIRGULA STRING CARACTERE
30
31 | %nonassoc LOWER_THAN_SENAO
32 | %nonassoc SENAO
33 | %left SOMASUB
34 | %left MULTDIV
35 | %%
36 | programa : declaracao_lista { fprintf(yyout,"%s", $1); }
37
38 | declaracao_lista : declaracao
   | {
39 |     char aux[10000];
40 |     strcpy(aux, $1);
41 |     int i;
42 |     char *result;
43 |     for(i = 0; i<current_tab_tipo ; i++){
44 |         if(tabela[i].flag_param == 1){
45 |             char temp_par[25];
46 |             char result_par[50];
47 |             char temp_vir[25];
48 |             char result_vir[20];
49
50 |             strcpy(temp_par, "(");
51 |             strcat(temp_par, tabela[i].tipo);
52 |             strcpy(result_par, temp_par);
53 |             if(strcmp(tabela[i].tipo, "char") == 0){
54 |                 strcat(result_par, tabela[i].nome);
55 |                 strcat(result_par, tabela[i].vet);
56 |             } else {
57 |                 strcat(result_par, "*");
58 |                 strcat(result_par, tabela[i].nome);
59 |             }
60 |             strcat(temp_par, tabela[i].nome);
61
62 |             strcpy(temp_vir, ",");
63 |             strcat(temp_vir, tabela[i].tipo);
64 |             strcpy(result_vir, temp_vir);
65 |             if(strcmp(tabela[i].tipo, "char") == 0){
66 |                 strcat(result_vir, tabela[i].nome);
67 |                 strcat(result_vir, tabela[i].vet);
68 |             } else {
69 |                 strcat(result_vir, "*");
70 |                 strcat(result_vir, tabela[i].nome);
71 |             }
72 |             strcat(temp_vir, tabela[i].nome);
73
74 |             result = repl_str(aux, temp_par, result_par);
75 |             result = repl_str(result, temp_vir, result_vir);
76
77 |             char temp[100];
78 |             strcpy(temp, result_vir);
79 |             strcat(temp, tabela[i].vet);
80 |             result = repl_str(result, temp, result_vir);
81
82 |             char temp2[100];
83 |             strcpy(temp2, result_par);
84 |             strcat(temp2, tabela[i].vet);
85 |             result = repl_str(result, temp2, result_par);
86
87 |             strcpy(aux, result);
88 |         }
89 |         if(tabela[i].flag_char == 1){
90 |             char temp_decl[40];
91 |             char result_decl[40];
92 |             char temp_gets[70];
93 |             char result_gets[90];
94
95 |             strcpy(temp_decl, tabela[i].nome);
96 |             strcat(temp_decl, "[50]");
97 |             strcpy(result_decl, tabela[i].nome);
98
99 |             strcpy(temp_gets, "fflush(stdin);\ngets(");
100 |             strcat(temp_gets, tabela[i].nome);
101 |             strcpy(result_gets, "scanf(\" %c\", &");
102 |             strcat(result_gets, tabela[i].nome);
103
104 |             result = repl_str(aux, temp_decl, result_decl);
105 |             result = repl_str(result, temp_gets, result_gets);
106 |             strcpy(aux, result);
107 |         }
108 |     }
109 |     $$ = strdup(aux);
110 |     free(result);
111 | }
112
113 | declaracao declaracao_lista
114 | {
115 |     char aux[10000];
116 |     strcpy(aux, $1);
117 |     strcat(aux, $2);
118 |     int i;
119 |     char *result;
120 |     for(i = 0; i<current_tab_tipo ; i++){
121 |         if(tabela[i].flag_param == 1){
122 |             char temp_par[25];
123 |             char result_par[50];
124 |             char temp_vir[25];
125 |             char result_vir[50];
126
127 |             strcpy(temp_par, "(");
128 |             strcat(temp_par, tabela[i].tipo);
129 |             strcpy(result_par, temp_par);
130 |             if(strcmp(tabela[i].tipo, "char") == 0){
131 |                 strcat(result_par, tabela[i].nome);
132 |                 strcat(result_par, tabela[i].vet);
133 |             } else {
134 |                 strcat(result_par, "*");
135 |                 strcat(result_par, tabela[i].nome);
136 |             }
137 |             strcat(temp_par, tabela[i].nome);
138
139 |             strcpy(temp_vir, ",");
140 |             strcat(temp_vir, tabela[i].tipo);
141 |             strcpy(result_vir, temp_vir);
142 |             if(strcmp(tabela[i].tipo, "char") == 0){
143 |                 strcat(result_vir, tabela[i].nome);
144 |                 strcat(result_vir, tabela[i].vet);
145 |             } else {
146 |                 strcat(result_vir, "*");
147 |                 strcat(result_vir, tabela[i].nome);
148 |             }

```

```

149         }
150         strcat(temp_vir, tabela[i].nome);
151
152         result = repl_str(aux, temp_par, result_par);
153         result = repl_str(result, temp_vir, result_vir);
154
155         char temp[100];
156         strcpy(temp, result_vir);
157         strcat(temp, tabela[i].vet);
158         result = repl_str(result, temp, result_vir);
159
160         char temp2[100];
161         strcpy(temp2, result_par);
162         strcat(temp2, tabela[i].vet);
163         result = repl_str(result, temp2, result_par);
164
165         strcpy(aux, result);
166     }
167     if(tabela[i].flag_char == 1){
168         char temp_decl[40];
169         char result_decl[40];
170         char temp_gets[70];
171         char result_gets[90];
172
173         strcpy(temp_decl, tabela[i].nome);
174         strcat(temp_decl, "[50]");
175         strcpy(result_decl, tabela[i].nome);
176
177         strcpy(temp_gets, "fflush(stdin);\ngets(");
178         strcat(temp_gets, tabela[i].nome);
179         strcpy(result_gets, "scanf(\" %c\", &");
180         strcat(result_gets, tabela[i].nome);
181
182         result = repl_str(aux, temp_decl, result_decl);
183         result = repl_str(result, temp_gets, result_gets);
184         strcpy(aux, result);
185     }
186     }
187     $$ = strdup(aux);
188     free(result);
189 }
190
191 declaracao : var_declaracao
192           | fun_declaracao
193
194 var_declaracao : tipo_especificador var_declaracao_meio
195               {
196                 char aux[100];
197                 strcpy(aux, $1);
198                 strcat(aux, $2);
199                 while(current_tab_tipo < current_tab_nome){
200                     strcpy(tabela[current_tab_tipo].tipo, $1);
201                     current_tab_tipo++;
202                 }
203                 $$ = strdup(aux);
204             }
205 | TEXTO decl_texto
206   {
207     char aux[100];
208     strcpy(aux, "char ");
209     strcat(aux, $2);
210     $$ = strdup(aux);
211   }
212 ;
213
214 decl_texto : id decl_texto_vet decl_texto_fat
215           {
216             char aux[100];
217             strcpy(aux, $1);
218             strcat(aux, $2);
219             strcpy(tabela[current_tab_nome].nome, $1);
220             strcat(aux, "[50]");
221             strcpy(tabela[current_tab_tipo].tipo, "char ");
222             strcpy(tabela[current_tab_nome].vet, $2);
223             strcat(tabela[current_tab_nome].vet, "[50]");
224             tabela[current_tab_nome].flag_string = 1;
225             current_tab_tipo++;
226             current_tab_nome++;
227             strcat(aux, $3);
228             $$ = strdup(aux);
229         }
230
231 decl_texto_vet : ABRECOLCHETE numint FECHACOLCHETE ATRIBUICAO var_vet_atr
232               {
233                 char aux[200];
234                 strcpy(aux, "[");
235                 strcat(aux, $2);
236                 strcat(aux, "] = ");
237                 strcat(aux, $5);
238                 $$ = strdup(aux);
239             }
240 | ABRECOLCHETE numint FECHACOLCHETE decl_texto_vet
241   {
242     char aux[100];
243     strcpy(aux, "[");
244     strcat(aux, $2);
245     strcat(aux, "]");
246     strcat(aux, $4);
247     $$ = strdup(aux);
248   }
249 | /*vazio*/ { $$ = strdup(""); }
250 ;
251 decl_texto_fat : VIRGULA decl_texto
252               {
253                 char aux[100];
254                 strcpy(aux, ", ");
255                 strcat(aux, $2);
256                 $$ = strdup(aux);
257             }
258 | /*vazio*/ { $$ = strdup(""); }
259 ;
260 var_declaracao_meio : id var_declaracao_vet var_declaracao_fat
261                    {
262                      char aux[100];
263                      strcpy(aux, $1);
264                      strcat(aux, $2);
265                      strcpy(tabela[current_tab_nome].nome, $1);
266                      strcpy(tabela[current_tab_nome].vet, $2);
267                      current_tab_nome++;
268                      strcat(aux, $3);
269                      $$ = strdup(aux);
270                  }
271 ;
272 var_declaracao_vet : ABRECOLCHETE numint FECHACOLCHETE ATRIBUICAO var_vet_atr

```



```

273     {
274         char aux[200];
275         strcpy(aux, "[");
276         strcat(aux, $2);
277         strcat(aux, "] = ");
278         strcat(aux, $5);
279         $$ = strdup(aux);
280     }
281 | ABRECOLCHETE numint FECHACOLCHETE var_declaracao_vet
282     {
283         char aux[100];
284         strcpy(aux, "[");
285         strcat(aux, $2);
286         strcat(aux, "]");
287         strcat(aux, $4);
288         $$ = strdup(aux);
289     }
290 | ATRIBUICAO expressao
291     {
292         char aux[100];
293         strcpy(aux, " = ");
294         strcat(aux, $2);
295         $$ = strdup(aux);
296     }
297 | /*vazio*/      { $$ = strdup(""); }
298 ;
299 var_declaracao_fat : VIRGULA var_declaracao_meio
300     {
301         char aux[100];
302         strcpy(aux, " ");
303         strcat(aux, $2);
304         $$ = strdup(aux);
305     }
306 | PONTOEVIRGULA { $$ = strdup(";\n"); }
307 ;
308 var_vet_atr : ABRECHAVE var_vet_dentro FECHACHAVE
309     {
310         char aux[100];
311         strcpy(aux, "(");
312         strcat(aux, $2);
313         strcat(aux, " ");
314         $$ = strdup(aux);
315     }
316 ;
317 var_vet_dentro : var_vet_atr VIRGULA var_vet_atr
318     {
319         char aux[100];
320         strcpy(aux, $1);
321         strcat(aux, ", ");
322         strcat(aux, $3);
323         $$ = strdup(aux);
324     }
325 | conteudo_vetor
326 ;
327 conteudo_vetor : numint vetor_int
328     {
329         char aux[100];
330         strcpy(aux, $1);
331         strcat(aux, $2);
332         $$ = strdup(aux);
333     }
334 | numreal vetor_real
335     {
336         char aux[100];
337         strcpy(aux, $1);
338         strcat(aux, $2);
339         $$ = strdup(aux);
340     }
341 | string vetor_string
342     {
343         char aux[100];
344         strcpy(aux, $1);
345         strcat(aux, $2);
346         $$ = strdup(aux);
347     }
348 | caractere vetor_caractere
349     {
350         char aux[100];
351         strcpy(aux, $1);
352         strcat(aux, $2);
353         $$ = strdup(aux);
354     }
355 ;
356 vetor_int : VIRGULA numint vetor_int
357     {
358         char aux[100];
359         strcpy(aux, " ");
360         strcat(aux, $2);
361         strcat(aux, $3);
362         $$ = strdup(aux);
363     }
364 | /*vazio*/      { $$ = strdup(""); }
365 ;
366 vetor_real : VIRGULA numreal vetor_real
367     {
368         char aux[100];
369         strcpy(aux, " ");
370         strcat(aux, $2);
371         strcat(aux, $3);
372         $$ = strdup(aux);
373     }
374 | /*vazio*/      { $$ = strdup(""); }
375 ;
376 vetor_string : VIRGULA string vetor_string
377     {
378         char aux[100];
379         strcpy(aux, " ");
380         strcat(aux, $2);
381         strcat(aux, $3);
382         $$ = strdup(aux);
383     }
384 | /*vazio*/      { $$ = strdup(""); }
385 ;
386 vetor_caractere : VIRGULA caractere vetor_caractere
387     {
388         char aux[100];
389         strcpy(aux, " ");
390         strcat(aux, $2);
391         strcat(aux, $3);
392         $$ = strdup(aux);
393     }
394 | /*vazio*/      { $$ = strdup(""); }
395 ;
396 tipo_especificador : INTEIRO { $$ = "int "; }

```

```

397 |         | REAL          { $$ = "float "; }
398 |         | SEMRETORNO    { $$ = "void "; }
399 |         | LOGICO        { $$ = "bool "; }
400 |
401 | id          : ID          ; { $$ = strdup(yytext); }
402 |
403 | numint     : NUMINT     { $$ = strdup(yytext); }
404 |
405 | numreal    : NUMREAL    { $$ = strdup(yytext); }
406 |
407 | string     : STRING     { $$ = strdup(yytext); }
408 |
409 | caractere  : CARACTERE  { $$ = strdup(yytext); }
410 |
411 | fun_declaracao : tipo_especificador id ABREPARENTESE params FECHAPARENTESE composto_decl
412 | {
413 |     char aux[5000];
414 |     strcpy(aux, $1);
415 |     strcat(aux, $2);
416 |     strcat(aux, "(");
417 |     strcat(aux, $4);
418 |     strcat(aux, ")");
419 |     strcat(aux, $6);
420 |     $$ = strdup(aux);
421 | }
422 | | tipo_especificador PRINCIPAL ABREPARENTESE params FECHAPARENTESE composto_decl
423 | {
424 |     char aux[5000];
425 |     strcpy(aux, $1);
426 |     strcat(aux, " main (");
427 |     strcat(aux, $4);
428 |     strcat(aux, ")");
429 |     strcat(aux, $6);
430 |     $$ = strdup(aux);
431 | }
432 | ;
433 | params      : param_lista
434 | | /*vazio*/ { $$ = strdup(""); }
435 | ;
436 | param_lista : param param_lista_fat
437 | {
438 |     char aux[100];
439 |     strcpy(aux, $1);
440 |     strcat(aux, $2);
441 |     $$ = strdup(aux);
442 | }
443 | ;
444 | param_lista_fat : VIRGULA param_lista
445 | {
446 |     char aux[100];
447 |     strcpy(aux, " ");
448 |     strcat(aux, $2);
449 |     $$ = strdup(aux);
450 | }
451 | | /*vazio*/ { $$ = strdup(""); }
452 | ;
453 | param       : tipo_especificador id
454 | {
455 |     char aux[100];
456 |     strcpy(aux, $1);
457 |     strcat(aux, $2);
458 |     $$ = strdup(aux);
459 | }
460 | | TEXTO id
461 | {
462 |     char aux[100];
463 |     strcpy(aux, "char ");
464 |     strcat(aux, $2);
465 |     $$ = strdup(aux);
466 | }
467 | ;
468 | composto_decl : ABRECHAVE dentro_funcao FECHACHAVE
469 | {
470 |     char aux[100];
471 |     strcpy(aux, "{ \n");
472 |     strcat(aux, $2);
473 |     strcat(aux, "\n");
474 |     $$ = strdup(aux);
475 | }
476 | ;
477 | dentro_funcao : var_declaracao dentro_funcao
478 | {
479 |     char aux[1000];
480 |     strcpy(aux, $1);
481 |     strcat(aux, $2);
482 |     $$ = strdup(aux);
483 | }
484 | | comando dentro_funcao
485 | {
486 |     char aux[5000];
487 |     strcpy(aux, $1);
488 |     strcat(aux, $2);
489 |     $$ = strdup(aux);
490 | }
491 | | /*vazio*/ { $$ = strdup(""); }
492 | ;
493 | comando     : expressao_decl
494 | | composto_decl
495 | | selecao_decl
496 | | iteracao_decl
497 | | para_decl
498 | | facaeng_decl
499 | | escolha_decl
500 | | leia_decl
501 | | escreva_decl
502 | | retorno_decl
503 | | interrompa_decl
504 | ;
505 | expressao_decl : expressao PONTOEVIRGULA
506 | {
507 |     char aux[1000];
508 |     strcpy(aux, $1);
509 |     strcat(aux, ";\n");
510 |     $$ = strdup(aux);
511 | }
512 | | PONTOEVIRGULA { $$ = strdup(";\n"); }
513 | ;
514 | selecao_decl  : SE ABREPARENTESE expressao FECHAPARENTESE comando %prec LOWER_THAN_SENAO
515 | {
516 |     char aux[1000];
517 |     strcpy(aux, "if (");
518 |     strcat(aux, $3);
519 |     strcat(aux, ")");

```

```

520         strcat(aux, $5);
521         $$ = strdup(aux);
522     }
523 | SE ABREPARENTESE expressao FECHAPARENTESE comando SENAO comando
524 {
525     char aux[1000];
526     strcpy(aux, "if (");
527     strcat(aux, $3);
528     strcat(aux, ")");
529     strcat(aux, $5);
530     strcat(aux, "else ");
531     strcat(aux, $7);
532     $$ = strdup(aux);
533 }
534 ;
535 iteracao_decl : ENQUANTO ABREPARENTESE expressao FECHAPARENTESE comando
536 {
537     char aux[1500];
538     strcpy(aux, "while (");
539     strcat(aux, $3);
540     strcat(aux, ")");
541     strcat(aux, $5);
542     $$ = strdup(aux);
543 }
544 ;
545 para_decl : PARA ABREPARENTESE id ATRIBUICAO expressao PONTOEVIRGULA
546           expressao PONTOEVIRGULA expressao_unaria FECHAPARENTESE comando
547 {
548     char aux[1000];
549     char operacao[20];
550     char passo[20];
551     int temp = atoi($9);
552     if(temp >= 0){
553         strcpy(operacao, $3);
554         strcat(operacao, "<=");
555         strcat(operacao, $7);
556         strcpy(passo, $3);
557         strcat(passo, "=");
558         strcat(passo, $3);
559         strcat(passo, "+");
560         strcat(passo, $9);
561     } else {
562         strcpy(operacao, $3);
563         strcat(operacao, ">=");
564         strcat(operacao, $7);
565         strcpy(passo, $3);
566         strcat(passo, "=");
567         strcat(passo, $3);
568         strcat(passo, $9);
569     }
570     strcpy(aux, "for (");
571     strcat(aux, $3);
572     strcat(aux, "= ");
573     strcat(aux, $5);
574     strcat(aux, "; ");
575     strcat(aux, operacao);
576     strcat(aux, "; ");
577     strcat(aux, passo);
578     strcat(aux, " ");
579     strcat(aux, $11);
580     $$ = strdup(aux);
581 }
582 ;
583 ;
584 facaeng_decl : FACA comando ENQUANTO ABREPARENTESE expressao FECHAPARENTESE PONTOEVIRGULA
585 {
586     char aux[1500];
587     strcpy(aux, "do");
588     strcat(aux, $2);
589     strcat(aux, "while (");
590     strcat(aux, $5);
591     strcat(aux, ");\n");
592     $$ = strdup(aux);
593 }
594 ;
595 escolha_decl : ESCOLHA ABREPARENTESE var FECHAPARENTESE ABRECHAVE
596             escolhas_dentro padrao_opc FECHACHAVE
597 {
598     char aux[1000];
599     strcpy(aux, "switch (");
600     strcat(aux, $3);
601     strcat(aux, "){\n");
602     strcat(aux, $6);
603     strcat(aux, $7);
604     strcat(aux, ")\n");
605     int i;
606     for(i = 0; i < current_tab_tipo; i++){
607         if(strcmp($3, tabela[i].nome) != NULL){
608             if(strcmp(tabela[i].tipo, "char") == 0){
609                 tabela[i].flag_char = 1;
610                 tabela[i].flag_string = 0;
611             }
612         }
613     }
614     $$ = strdup(aux);
615 }
616 ;
617 ;
618 ;
619 leia_decl : LEIA ABREPARENTESE string VIRGULA var FECHAPARENTESE PONTOEVIRGULA
620 {
621     char *result;
622     char aux[400];
623     char aux2[400];
624     if(strcmp($3, "%texto") != NULL){
625         strcpy(aux, "fflush(stdin);\ngets(");
626         strcat(aux, $5);
627         strcat(aux, ");\n");
628     } else {
629         result = repl_str($3, "%inteiro", "%d");
630         result = repl_str(result, "%real", "%f");
631         strcpy(aux, "scanf(");
632         strcat(aux, result);
633         strcat(aux, ");\n");
634     }
635     $$ = strdup(aux);
636     free(result);
637 }
638 ;
639 escreva_decl : ESCRIVA ABREPARENTESE string escreva_fator FECHAPARENTESE PONTOEVIRGULA
640 {
641     char *result;
642     char aux[400];
643     result = repl_str($3, "%inteiro", "%d");
644     result = repl_str(result, "%real", "%f");

```

```

645         result = repl_str(result,"%texto","%s");
646         result = repl_str(result,"%5real","%.5f");
647         strcpy(aux, "printf (");
648         strcat(aux, result);
649         strcat(aux, $4);
650         strcat(aux, ");\n");
651         $$ = strdup(aux);
652         free(result);
653     }
654     maiuscula_decl : MAIUSCULA ABREPARENTESE char_fator FECHAPARENTESE
655     {
656         char aux[200];
657         strcpy(aux, "toupper (");
658         strcat(aux, $3);
659         strcat(aux, ");");
660         $$ = strdup(aux);
661     }
662     concatena_decl : CONCATENA ABREPARENTESE concatena_fator VIRGULA concatena_fator FECHAPARENTESE
663     {
664         char aux[200];
665         strcpy(aux, "strcat (");
666         strcat(aux, $3);
667         strcat(aux, ", ");
668         strcat(aux, $5);
669         strcat(aux, ");");
670         $$ = strdup(aux);
671     }
672     interrompa_decl : INTERROMPA PONTOEVIRGULA { $$ = strdup("break;\n"); }
673     concatena_fator : string
674     | var
675     | retorno_decl
676     | escreva_fator
677     | /*vazio*/ { $$ = strdup(""); }
678     | char_fator
679     | escolhas_dentro
680     ;
681     retorno_decl : RETORNE PONTOEVIRGULA { $$ = strdup("return;\n"); }
682     | RETORNE expressao PONTOEVIRGULA
683     {
684         char aux[50];
685         strcpy(aux, "return ");
686         strcat(aux, $2);
687         strcat(aux, ");\n");
688         $$ = strdup(aux);
689     }
690     escreva_fator : VIRGULA expressao escreva_fator
691     {
692         char aux[100];
693         strcpy(aux, ", ");
694         strcat(aux, $2);
695         strcat(aux, $3);
696         $$ = strdup(aux);
697     }
698     | /*vazio*/ { $$ = strdup(""); }
699     ;
700     char_fator : var
701     | caractere
702     ;
703     escolhas_dentro : CASO escolha_simples escolhas_dentro_mais
704     | DOISPONTOS comando_caso escolhas_dentro
705     {
706         char aux[400];
707         strcpy(aux, "case ");
708         strcat(aux, $2);
709         strcat(aux, $3);
710         strcat(aux, "\n");
711         strcat(aux, $5);
712         strcat(aux, "break;\n");
713         strcat(aux, $6);
714         $$ = strdup(aux);
715     }
716     | /*vazio*/ { $$ = strdup(""); }
717     ;
718     escolhas_dentro_mais : VIRGULA escolha_simples escolhas_dentro_mais
719     {
720         char aux[100];
721         strcpy(aux, ":ncase ");
722         strcat(aux, $2);
723         strcat(aux, $3);
724         $$ = strdup(aux);
725     }
726     | /*vazio*/ { $$ = strdup(""); }
727     ;
728     comando_caso : /*vazio*/ { $$ = strdup(""); }
729     | comando
730     ;
731     padrao_opc : PADRAO DOISPONTOS comando
732     {
733         char aux[400];
734         strcpy(aux, "default :\n");
735         strcat(aux, $3);
736         $$ = strdup(aux);
737     }
738     | /*vazio*/ { $$ = strdup(""); }
739     ;
740     expressao : var ATRIBUICAO expressao
741     {
742         char aux[300];
743         strcpy(aux, $1);
744         strcat(aux, " = ");
745         strcat(aux, $3);
746         int i;
747         for (i = 0; i < current_tab_tipo; i++) {
748             if (strcmp($1, tabela[i].nome) == 0) {
749                 if ((tabela[i].flag_string == 1) && (tabela[i].flag_char != 1)) {
750                     strcpy(aux, "strcpy(");
751                     strcat(aux, $1);
752                     strcat(aux, ", ");
753                     strcat(aux, $3);
754                     strcat(aux, ");");
755                 }
756             }
757         }
758         $$ = strdup(aux);
759     }
760     | expressao_logica
761     ;
762     var : id var_fat
763     {
764         char aux[50];
765         strcpy(aux, $1);
766         strcat(aux, $2);
767         $$ = strdup(aux);
768     }

```

```

769 |
770 | var_fat      : ABRECOLCHETE expressao FECHACOLCHETE var_fat
771 | {
772 |     char aux[100];
773 |     strcpy(aux, "");
774 |     strcat(aux, $2);
775 |     strcat(aux, " ");
776 |     strcat(aux, $4);
777 |     $$ = strdup(aux);
778 | }
779 | /*vazio*/   { $$ = strdup(""); }
780 |
781 | expressao_logica : OU termo_logico expressao_logica_fat
782 | {
783 |     char aux[200];
784 |     strcpy(aux, "||");
785 |     strcat(aux, $2);
786 |     strcat(aux, $3);
787 |     $$ = strdup(aux);
788 | }
789 |
790 | | termo_logico expressao_logica_fat
791 | {
792 |     char aux[200];
793 |     strcpy(aux, $1);
794 |     strcat(aux, $2);
795 |     $$ = strdup(aux);
796 | }
797 |
798 | expressao_logica_fat : OU termo_logico expressao_logica_fat
799 | {
800 |     char aux[200];
801 |     strcpy(aux, "||");
802 |     strcat(aux, $2);
803 |     strcat(aux, $3);
804 |     $$ = strdup(aux);
805 | }
806 | /*vazio*/   { $$ = strdup(""); }
807 |
808 | termo_logico : expressao_simples termo_logico_fat
809 | {
810 |     char aux[200];
811 |     strcpy(aux, $1);
812 |     int i;
813 |     for(i = 0; i < current_tab_tipo; i++){
814 |         if(strstr($1, tabela[i].nome) != NULL){
815 |             if(strstr($1, "\'") != NULL){
816 |                 if(strstr($1, "\"" == NULL){
817 |                     if((strstr($1, "!=") != NULL || strstr($1, "==") != NULL){
818 |                         tabela[i].flag_char = 1;
819 |                         tabela[i].flag_string = 0;
820 |                     }
821 |                 }
822 |             }
823 |         }
824 |     }
825 |     strcat(aux, $2);
826 |     $$ = strdup(aux);
827 | }
828 |
829 | termo_logico_fat : E expressao_simples termo_logico_fat
830 | {
831 |     char aux[200];
832 |     strcpy(aux, "&&");
833 |     strcat(aux, $2);
834 |     strcat(aux, $3);
835 |     $$ = strdup(aux);
836 | }
837 | /*vazio*/   { $$ = strdup(""); }
838 |
839 | expressao_simples : expressao_soma expressao_simples_fat
840 | {
841 |     char aux[200];
842 |     strcpy(aux, $1);
843 |     strcat(aux, $2);
844 |     int i;
845 |     for(i = 0; i < current_tab_tipo; i++){
846 |         if(strcmp($1, tabela[i].nome) == 0){
847 |             if((tabela[i].flag_string == 1) && (tabela[i].flag_char != 1)){
848 |                 if(strstr($2, "=") != NULL){
849 |                     if(strstr($2, "\'") == NULL){
850 |                         char *result;
851 |                         strcpy(aux, "strcmp(");
852 |                         strcat(aux, $1);
853 |                         result = repl_str($2, "=", "");
854 |                         strcat(aux, ", ");
855 |                         strcat(aux, result);
856 |                         strcat(aux, ") == 0");
857 |                     }
858 |                 }
859 |             }
860 |         }
861 |     }
862 |     $$ = strdup(aux);
863 | }
864 |
865 | | EXCLAMACAO expressao_soma expressao_simples_fat
866 | {
867 |     char aux[200];
868 |     strcpy(aux, "!");
869 |     strcat(aux, $2);
870 |     strcat(aux, $3);
871 |     $$ = strdup(aux);
872 | }
873 |
874 | | VERDADEIRO { $$ = strdup("true"); }
875 | | FALSO      { $$ = strdup("false"); }
876 |
877 | relacional : RELACIONAL { $$ = strdup(yytext); }
878 |
879 | expressao_simples_fat : relacional expressao_soma
880 | {
881 |     char aux[200];
882 |     strcpy(aux, $1);
883 |     strcat(aux, $2);
884 |     $$ = strdup(aux);
885 | }
886 | /*vazio*/   { $$ = strdup(""); }
887 |
888 | somasub : SOMASUB { $$ = strdup(yytext); }
889 |
890 | expressao_soma : somasub termo expressao_soma_fat
891 | {
892 |     char aux[200];
893 |     strcpy(aux, $1);
894 |     strcat(aux, $2);

```

```

893 |                                     strcat(aux, $3);
894 |                                     $$ = strdup(aux);
895 |                                 }
896 |         | termo expressao_soma_fat
897 |         {
898 |             char aux[200];
899 |             strcpy(aux, $1);
900 |             strcat(aux, $2);
901 |             $$ = strdup(aux);
902 |         }
903 |     };
904 | expressao_soma_fat : somasub termo expressao_soma_fat
905 |     {
906 |         char aux[200];
907 |         strcpy(aux, $1);
908 |         strcat(aux, $2);
909 |         strcat(aux, $3);
910 |         $$ = strdup(aux);
911 |     }
912 |     | /*vazio*/      { $$ = strdup(""); }
913 |     ;
914 | termo : fator termo_fat
915 |     {
916 |         char aux[200];
917 |         strcpy(aux, $1);
918 |         strcat(aux, $2);
919 |         $$ = strdup(aux);
920 |     }
921 |     ;
922 | multdiv : MULTDIV      { $$ = strdup(yytext); }
923 |     ;
924 | termo_fat : multdiv fator termo_fat
925 |     {
926 |         char aux[200];
927 |         strcpy(aux, $1);
928 |         strcat(aux, $2);
929 |         $$ = strdup(aux);
930 |     }
931 |     | /*vazio*/      { $$ = strdup(""); }
932 |     ;
933 |     ;
934 | fator : ABREPARENTESE expressao FECHAPARENTESE
935 |     {
936 |         char aux[200];
937 |         strcpy(aux, "(");
938 |         strcat(aux, $2);
939 |         strcat(aux, ")");
940 |         $$ = strdup(aux);
941 |     }
942 |     | var
943 |     | ativacao
944 |     | numreal
945 |     | numint
946 |     | caractere
947 |     | string
948 |     | raizq_decl
949 |     | potencia_decl
950 |     | tamanhotexto_decl
951 |     | maiuscula_decl
952 |     | concatena_decl
953 |     ;
954 | expressao_unaria : numint
955 |     | somasub numint      {
956 |         char aux[30];
957 |         strcpy(aux, $1);
958 |         strcat(aux, $2);
959 |         $$ = strdup(aux);
960 |     }
961 |     ;
962 | escolha_simples : expressao_unaria
963 |     | caractere
964 |     ;
965 | raizq_decl : RAIZQUADRADA ABREPARENTESE expressao_soma FECHAPARENTESE
966 |     {
967 |         char aux[150];
968 |         strcpy(aux, "sqrt (");
969 |         strcat(aux, $3);
970 |         strcat(aux, ")");
971 |         $$ = strdup(aux);
972 |     }
973 |     ;
974 | potencia_decl : POTENCIA ABREPARENTESE expressao_soma VIRGULA expressao_soma FECHAPARENTESE
975 |     {
976 |         char aux[100];
977 |         strcpy(aux, "pow (");
978 |         strcat(aux, $3);
979 |         strcat(aux, ", ");
980 |         strcat(aux, $5);
981 |         strcat(aux, ")");
982 |         $$ = strdup(aux);
983 |     }
984 |     ;
985 | tamanhotexto_decl : TAMANHOTEXTO ABREPARENTESE texto_fator FECHAPARENTESE
986 |     {
987 |         char aux[100];
988 |         strcpy(aux, "strlen (");
989 |         strcat(aux, $3);
990 |         strcat(aux, ")");
991 |         $$ = strdup(aux);
992 |     }
993 |     ;
994 | texto_fator : var
995 |     | string
996 |     ;
997 | ativacao : id ABREPARENTESE args FECHAPARENTESE
998 |     {
999 |         char aux[100];
1000 |         strcpy(aux, $1);
1001 |         strcat(aux, "(");
1002 |         int i;
1003 |         for (i = 0; i < current_tab.tipo; i++) {
1004 |             if (strstr($3, tabela[i].nome) != NULL) {
1005 |                 if (strstr(tabela[i].vet, "(") != NULL) {
1006 |                     tabela[i].flag_param = 1;
1007 |                 }
1008 |             }
1009 |         }
1010 |         strcat(aux, $3);
1011 |         strcat(aux, ")");
1012 |         $$ = strdup(aux);
1013 |     }
1014 |     ;
1015 | args : arg_lista

```

```

1016 |         | /*vazio*/      { $$ = strdup(""); }
1017 |         ;
1018 | arg_lista : expressao args_fat
1019 |         {
1020 |             char aux[100];
1021 |             strcpy(aux, $1);
1022 |             strcat(aux, $2);
1023 |             $$ = strdup(aux);
1024 |         }
1025 |         ;
1026 | args_fat  : VIRGULA expressao args_fat
1027 |         {
1028 |             char aux [100];
1029 |             strcpy(aux, " ");
1030 |             strcat(aux, $2);
1031 |             strcat(aux, $3);
1032 |             $$ = strdup(aux);
1033 |         }
1034 |         | /*vazio*/      { $$ = strdup(""); }
1035 |         ;
1036 | %%
1037 | void yyerror (char *s) {
1038 |     fprintf(stderr, "%s\n", s);
1039 | }
1040 |
1041 | int main(int argc, char *argv[]){
1042 |     yyout = fopen(argv[2], "w");
1043 |     fprintf(yyout, "#include <stdio.h>\n");
1044 |     fprintf(yyout, "#include <stdlib.h>\n");
1045 |     fprintf(yyout, "#include <stdbool.h>\n");
1046 |     fprintf(yyout, "#include <math.h>\n");
1047 |     fprintf(yyout, "#include <string.h>\n");
1048 |     fprintf(yyout, "\n");
1049 |     yyin = fopen(argv[1], "r");
1050 |     yyparse();
1051 |     return 0;
1052 | }

```