Escola Politécnica da Universidade de São Paulo

PCS3216 Sistemas de Programação
Prova 1

Decomposição de texto

(problema 5)

Otávio Henrique Monteiro 10774159

Descrição conceitual

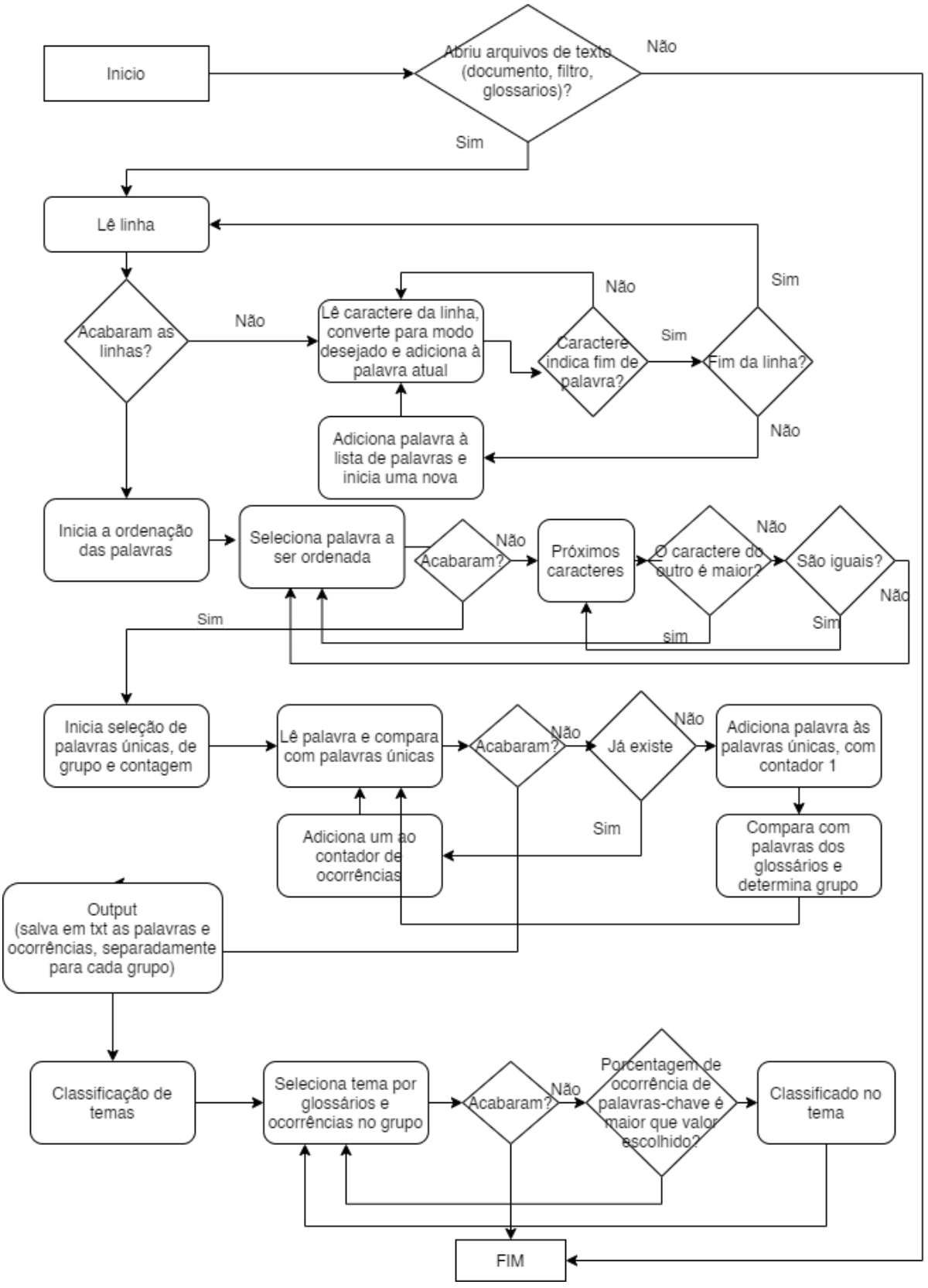
O software desenvolvido tem como objetivo principal decompor um texto em suas palavras e, por meio da análise da a ocorrência dessas, separar e classificar em temas pré-determinados. Foram criadas listas de palavras referentes à tarefa de filtrar, selecionar e classificar o texto, sendo que essas foram montadas a partir de análises de textos processados pelo próprio programa ao longo de seu desenvolvimento.

Escrito em C++, as palavras são *structs* que contém informação dos caracteres que a compõem, do número de ocorrências ao longo do texto selecionado e a qual grupo de palavra pertence; este último é referente ao conceito de palavras-chave e temas.

Em suma, as atividades realizadas pelo software são: leitura de palavras de textos, filtro e glossários; ordenação de lista de palavras e contagem de ocorrências; escrita da saída desejada, incluindo a classificação do tema do texto original.

Os códigos, executáveis e textos podem ser encontrados no repositório do GitHub. O código em C++ encontra-se também ao fim do arquivo.

Pseudocódigo/diagrama de blocos



Modelagem do programa

O projeto é baseado em motores de eventos, notando-se especialmente a porção de leitura e transformação de caracteres e palavras e a porção de ordenação. Ilustra-se aqui o modelo de simulação guiada por eventos para esse programa.

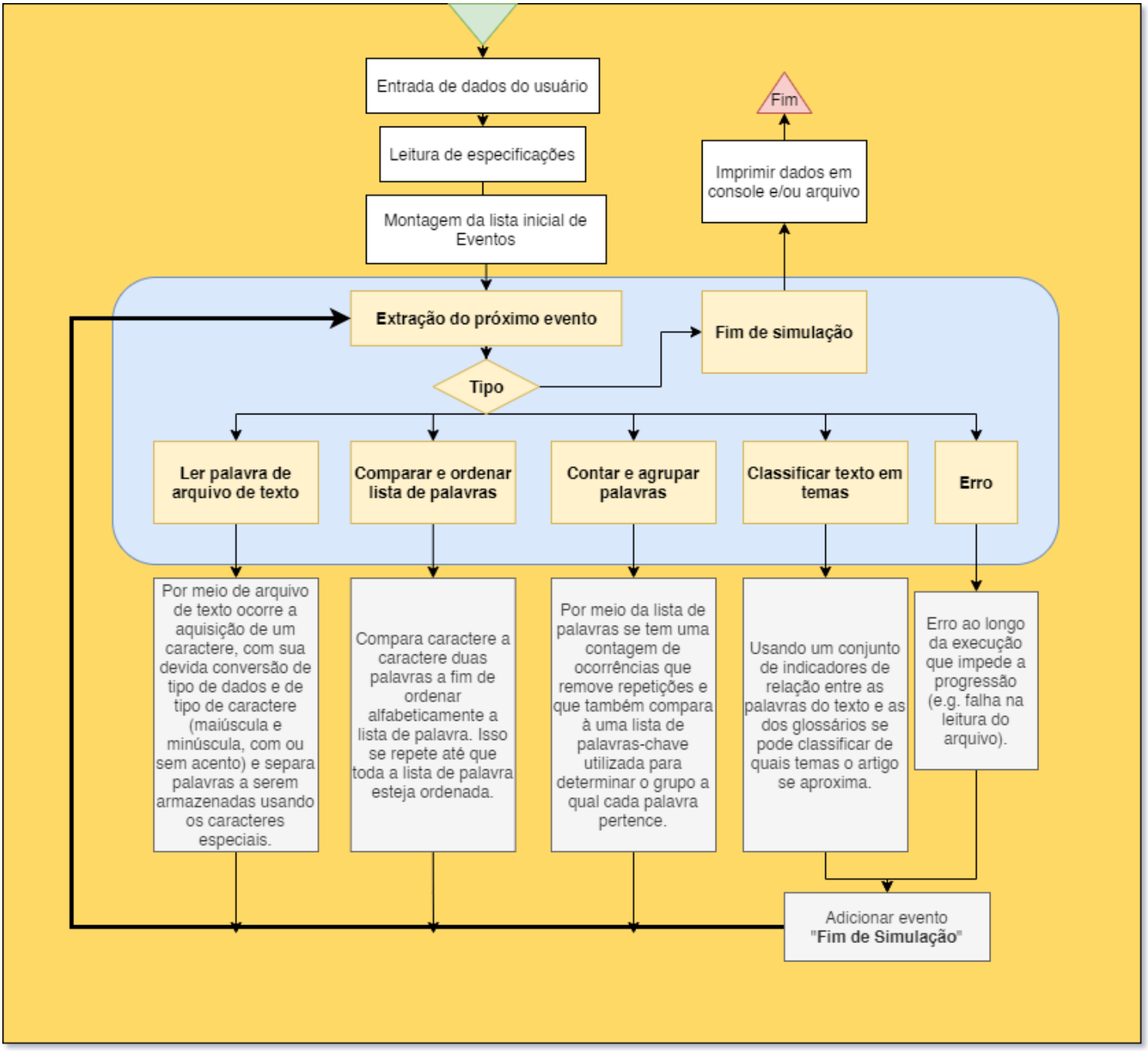


Tabela de eventos

A tabela a seguir apresenta os tipos de eventos e sua correspondente reação em termos gerais de comportamento e de próximos eventos.

Eventos	Reações
Ler palavra de arquivo de texto	Lê palavra do texto e retorna ao motor de eventos o seu próprio evento se houverem outras palavras a serem lidas, do contrário retorna evento de comparar e ordenar palavras.
Comparar e ordenar lista de palavras	Ordena lista de palavras por meio da comparação de caracteres e ao término da ordenação retorna ao motor de eventos o evento de contar e agrupar palavras.
Contar e agrupar palavras	Conta a ocorrência de palavras e compara com as diferentes listas de palavras-chave (glossários). Enquanto não satisfeito o agrupamento para todos os grupos de glossários, o evento se mantém no motor de eventos. Posteriormente retorna evento de Classificar texto em temas.
Classificar texto em temas	Classifica para todos os temas e retorna ao motor de eventos o evento de Fim de Simulação.
Erro	Quando ocorre um erro na execução o sistema para seu funcionamento e retorna ao motor de eventos o evento de Fim de Simulação.

Testagem

Tendo funcional um programa de decomposição de texto com contagem e ordenamento, os testes realizados ocorreram inicialmente para 12 textos, de três temas: “computação quântica”, “biotecnologia” e “psicologia”.

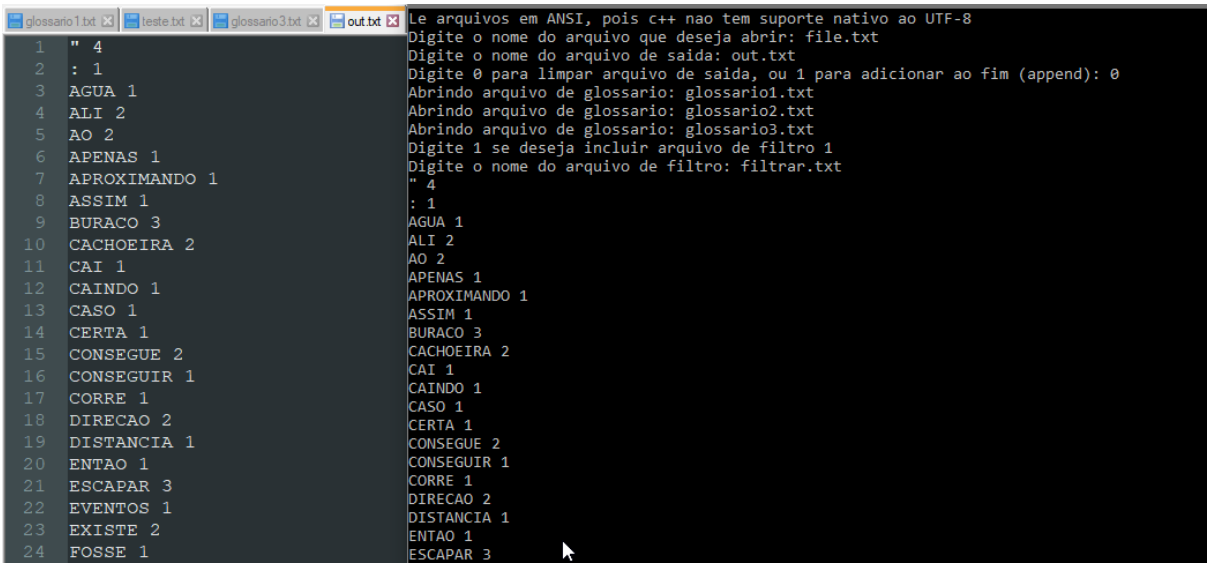
Desses textos foram retiradas informações sobre palavras e caracteres que aparecem com frequência ao texto, mas são de menor importância relativa ao tema do artigo; assim foi criado um arquivo contendo “palavras” a serem filtradas dos demais textos. O método particular utilizado para realizar essa filtragem foi classificar as palavras existentes em ambas listas de palavras (filtro e texto a ser filtrado) como um grupo extra a ser ignorado no processamento.

Foram montados também três glossários, utilizando termos específicos dos temas que notavelmente, por meio da análise criada pelo programa, tem ocorrência frequente em artigos de seu tema.

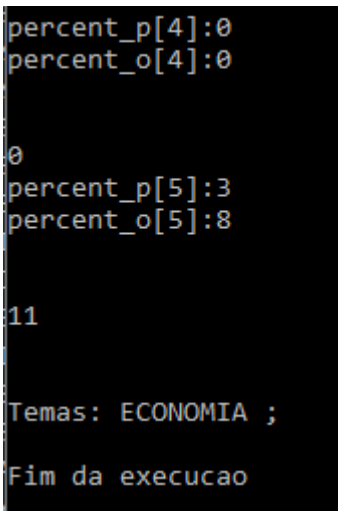
É esperado que esses glossários sejam úteis para classificar qual o tema desses próprios artigos ou de um artigo suficientemente próximo de um ou mais dos temas. Nota-se que a quantidade de termos dos glossários é pequena e que variações na palavra (e.g. em conjugação) impede a detecção do tema.

Com novos textos de temas mais variados, houve a necessidade de ampliar os glossários para variações e novos termos. Assim, a classificação de temas depende da proximidade do texto com o tema tratado pelos artigos que foram as fontes dos glossários. Posteriormente foram adicionados novos glossários (e.g. tema “política”), para a classificação de diferentes textos, demonstrando a capacidade de classificação em mais de um tema. Foram utilizados para testes 8 textos com variados temas, dentre os três anteriores, os novos e temas sem glossário.

A classificação se deu de diferentes formas, à amenizar problemas como poucas palavras no glossário e múltiplas ocorrências de poucas palavras, mas que são de grande importância para o tema. Foram analisados para os textos o número de ocorrências de palavras dentro dos temas em comparação com o número de palavras no total, o número de palavras que fazem parte do tema em relação com o número de palavras únicas filtradas e também a relação do número de palavras que fazem parte do glossário e que aparecem no texto (esta última é relevante para glossários reduzidos).



Na figura acima, as entradas de usuário, saídas pelo console e arquivo final do texto fornecido no enunciado estão visíveis.



Na figura ao lado é possível ver os indicadores previamente comentados para dois diferentes temas, "Política" e “Economia” . O artigo tratava do uso de criptomoedas na economia, e seus indicadores classificaram corretamente o tema do texto tratado.
Os demais temas também foram avaliados pelo classificador, que pode ter como classificação um conjunto de temas, se satisfeitas as condições para cada. Variadas condições foram testadas, utilizando textos que foram arcabouço para a construção de glossários e textos dentro do mesmo tema que não tinham seus termos selecionados para tal. A condição final para classificar textos multidisciplinares foi: se uma das duas condições for satisfeita considera-se que o texto faz parte do tema analisado.

- Se 3% ou mais das palavras totais do texto estão presentes no glossário
- Se 20% ou mais das palavras do glossário se encontram no texto

Análise comentada dos resultados

É uma maneira rústica de classificação que pode apresentar falsos positivos para temas similares e falsos negativos por conta do uso dos termos. Por exemplo, em um dos testes com artigos diversos, o classificador deu o falso-positivo do tema “computação quântica” por se tratar de um artigo sobre inteligência artificial, por se tratar de um assunto de computação. Com relação à falsos-negativos, podemos salientar que diferenças em termos como "comportamento", "comportamental" e "comportamentais" podem fazer com que o classificador não identifique o tema. Para corrigir (ou, ao menos, amenizar) o problema foram ampliados os glossários com variações para termos existentes, além da adição de novos termos similares da área.

Código (C++)

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <list>
#include <deque>
#include <stdio.h>

using namespace std;
int caracterEspecial(int l);

struct Palavras
{
    std::string word;
    int occurency;
    int group;
};

int main(){
    // Apresenta��o
    cout << "Le arquivos em ANSI, pois c++ nao tem suporte nativo ao UTF-8" << endl;

    //Entradas de usuario
    cout << "Digite o nome do arquivo que deseja abrir: ";
    string nome_arquivo;
    cin >> nome_arquivo;
    cout << "Digite o nome do arquivo de saida: ";
    string nome_arquivo_out;
    cin >> nome_arquivo_out;
    cout << "Digite 0 para limpar arquivo de saida, ou 1 para adicionar ao fim (append): ";
    int modo_arquivo_saida = 0;
    cin >> modo_arquivo_saida;


    //Listas de palavras, palavras-chave e filtro
    std::list<std::string> palavras;
    std::list<std::string> filtrar;

    //Entrada de gloss�rio (key-words)
    int n_kw = 5; //N� de grupos
```

```
    std::list<std::string>* kw = NULL;
    //std::string* glossarios = NULL;
    kw = new std::list<std::string>[n_kw + 1];
    //glossarios = new std::string[n_kw + 1];

    ////Metodo de entrada de glossario pelo usuario
    //const char parada = '.';
    //string entrada;
    //cout << endl << "Entrada de palavras do glossario (em maiusculo, sem
acentos; '.' para parar): " << endl;
    //while (parada != entrada[0])
    //{
    //  cin >> entrada;
    //  kw1.push_back(entrada);
    //}
    //kw1.pop_back(); //removendo o indicador de parada

    //Palavras-chave (glossario) precisam estar no padrao usado pelo programa

    for (int i = 1; i <= n_kw; i++)
    {
        string caminho_glossario = "glossario"+ to_string(i)+ ".txt";
        cout << "Abrindo arquivo de glossario: " << caminho_glossario << endl;
        string keyword;
        ifstream g; g.open(caminho_glossario.c_str(), ios::in);
        while (getline(g, keyword)) {
            kw[i].push_back(keyword);
        }
        g.close();
        //kw[i] =
    }

    //Entrada de filtro
    cout << "Digite 1 se deseja incluir arquivo de filtro ";
    int incluirFiltro = 0;
    cin >> incluirFiltro;
    if (incluirFiltro == 1) {
        cout << "Digite o nome do arquivo de filtro: ";
        string nome_arquivo_filtro;
        cin >> nome_arquivo_filtro;

        ifstream filtro;
```

```
        filtro.open(nome_arquivo_filtro.c_str(), ios::in);
        if (filtro.is_open()) {
            string wordToFilter;
            while (getline(filtro, wordToFilter)) {
                filtrar.push_back(wordToFilter);
            }
        }
    }
    //Para filtrar, iremos associar as palavras do filtro com um grupo de
valor superior ao numero de grupos de palavras-chave
    //palavras filtro tem valor de grupo n_kw + 1




    //Preparando arquivo
    ifstream arquivo;
    arquivo.open(nome_arquivo.c_str(),ios::in);

    if (arquivo.is_open()){   //se o arquivo estiver aberto executa o codigo
        //cout << "ABRIU ARQUIVO" << endl;
        //No inicio pega uma linha, passa todos os caracteres para o tipo
desejado (maiusculo sem acentos){n□o distingue palavras com e sem acentos,
e.g. "nos" e "n□s"}
        string linhaAtual;

        while(getline(arquivo, linhaAtual)){  //le uma linha do arquivo e poe
na string tp
            std::list<char> letras;
            //cout << "Linha: " <<linhaAtual<< endl;

            for (unsigned int i = 0; i < linhaAtual.length(); i++)
            {
                unsigned char l = linhaAtual[i];
                //maiusculas
                if (l >= 65 && l <= 90) { letras.push_back(l); }
                //minusculas para maiusculas
                else if (l >= 97 && l <= 122) { letras.push_back(l+65-97); }
                //caracteres especiais e acentos
                else { letras.push_back(caracterEspecial(l)); }
            }

            //Passa de letra em letra construindo uma string
```

```
            //quando achar separador (caracter especial, doferente de letras
comuns ou numeros)
            //coloca a palavra na lista de palavras
            string palavraAtual;
            std::list<char>::iterator it_letras;

            for (it_letras = letras.begin(); it_letras != letras.end();
++it_letras) {
                //avalia fim da palavra
                bool fimDaPalavra;
                int l = *it_letras; //Cast de char para int

                switch (l)
                {
                case 48 ... 57: //numeros
                    fimDaPalavra = false;
                    break;
                case 65 ... 90: //letras
                    fimDaPalavra = false;
                    break;
                default:
                    fimDaPalavra = true;
                    break;
                }


                if (fimDaPalavra) {
                    //coloca a palavra completa na lista de palavras
                    palavras.push_back(palavraAtual);
                    //reinicia a palavra atual
                    palavraAtual = "";
                    //palavra atual como caracter especial
                    palavraAtual.push_back(*it_letras);
                    palavras.push_back(palavraAtual);
                    palavraAtual = "";
                    //cout << "COMPLETOU UMA PALAVRA" << endl;
                }
                else {
                    palavraAtual.push_back(*it_letras); //poe caracter na
palavra
                }

            }
```

```
            //Ultima palavra antes do fim da linha
            palavras.push_back(palavraAtual);

        }


      arquivo.close();   //fecha arquivo
    }

    //Ordenar a lista antes da contagem
    palavras.sort();

    std::list<std::string>::iterator it_palavras;
    std::deque<Palavras> p_unicas;
    std::deque<int> ocorrencias;



    for (it_palavras = palavras.begin(); it_palavras != palavras.end();
++it_palavras) {

        bool jaExiste = false;

        std::deque<Palavras>::iterator itpu; //iterador de palavras unicas
        for (itpu=p_unicas.begin(); itpu != p_unicas.end(); ++itpu)
        {
            if ((*itpu).word == *it_palavras) {//Testa se as palavras s□o
iguais. Precisa ser m□todo pr□prio de compara□□o?
                jaExiste = true;
                (*itpu).occurency++;
            }
        }

        if (!jaExiste){//Se ainda nao apareceu, cria uma posicao e poe 1 no
contador de ocorrencias
            //struct Palavras* p = (struct Palavras*) malloc(sizeof(struct
Palavras));
            Palavras p;
            p.word = *it_palavras;
            p.occurency = 1;
            p.group = 0;
            p_unicas.push_back(p);
        }
```

```
    }


    //Seleciona grupos
    std::deque<Palavras>::iterator itpu; //iterador de palavras unicas
    for (itpu = p_unicas.begin(); itpu != p_unicas.end(); ++itpu)
    {
        //Palavras chave
        for (int i = 1; i <= n_kw; i++)
        {
            for (std::list<std::string>::iterator itkw = kw[i].begin(); itkw
!= kw[i].end(); ++itkw)
            {
                if ((*itpu).word == *itkw) { (*itpu).group = i; }
            }
        }


        //Palavras a serem filtradas
        for (std::list<std::string>::iterator it_filtro = filtrar.begin();
it_filtro != filtrar.end(); ++it_filtro)
        {
            if ((*itpu).word == *it_filtro) { (*itpu).group = n_kw + 1; }
        }
    }




    //OUTPUT
    ofstream out_f;
    if (modo_arquivo_saida == 1) { out_f.open(nome_arquivo_out.c_str(),
ios::app); }
    else                         { out_f.open(nome_arquivo_out.c_str(),
ios::out); }

    //Contadores de palavras de todos os temas e de numero de ocorrencias
dentro do tema
    int* n_palavras = NULL;
    n_palavras = new int[n_kw+1];
    int* n_ocorrencias = NULL;
    n_ocorrencias = new int[n_kw + 1];
```

```
    int palavras_totais, ocorrencias_totais;
    //zerando arrays
    for (int i=0; i < n_kw+1; i++){
        n_palavras[i] = 0;
        n_ocorrencias[i] = 0;
    }


    for (int i = 0; i <= n_kw; i++)
    {
        //out_f << endl << "Grupo " << i << endl;
        //out_f << "Palavras ; Occorrencias:" << endl << endl;
        std::deque<Palavras>::iterator itpu; //iterador de palavras unicas
        for (itpu = p_unicas.begin(); itpu != p_unicas.end(); ++itpu)
        {
            if (i == (*itpu).group) {
                out_f << (*itpu).word <<" "<< (*itpu).occurency << endl;
                cout << (*itpu).word << " "<< (*itpu).occurency << endl;
                n_palavras[i] += 1;
                n_ocorrencias[i] += (*itpu).occurency;
            }
        }
        out_f << endl << endl;//Separacao

        palavras_totais     += n_palavras[i];
        ocorrencias_totais  += n_ocorrencias[i];
    }

    out_f.close();

    //Avaliando classifica��o
    float* percent_p = NULL;
    float* percent_o = NULL;
    percent_p = new float[n_kw + 1];
    percent_o = new float[n_kw + 1];
    string temas;

    //Classificando:
    for (int i = 1; i <= n_kw; i++)
    {
        percent_p[i] = n_palavras[i] * 100 / palavras_totais;
        percent_o[i] = n_ocorrencias[i] * 100 / ocorrencias_totais;
```

```
        float percent_g = n_palavras[i] * 100 / kw[i].size();//Porcentagem de
palavras do glossario que apareceram no texto

        cout << "percent_p[" << i << "]:" << percent_p[i] << endl;
        cout << "percent_o[" << i << "]:" << percent_o[i] << endl;
        cout << endl << endl << percent_g << endl;

        if (percent_o[i] > 3.0 || percent_g > 20) {
            std::list<std::string>::iterator itkw = kw[i].begin();
            temas += *itkw + "; ";
        }
    }


    cout << endl << endl << "Temas: " << temas << endl;

    //Limpando memoria
    delete[] n_ocorrencias; n_ocorrencias = NULL;
    delete[] n_palavras; n_palavras = NULL;

    //Fechamento
    cout << endl  <<"Fim da execucao"<< endl;
    int close;
    cin >> close;

}

int caracterEspecial(int l) {
    int out;
    //cout << "l: " <<l<< endl;
    switch (l)
    {
        //A
        case 192 ... 197:
            out = 65;
            break;
        case 225 ... 229:
            out = 65;
            break;
        //E
        case 200 ... 203:
            out = 69;
            break;
```

```
        case 232 ... 235:
            out = 69;
            break;
        //I
        case 204 ... 207:
            out = 73;
            break;
        case 236 ... 239:
            out = 73;
            break;
        //O
        case 210 ... 214:
            out = 79;
            break;
        case 242 ... 246:
            out = 79;
            break;
        //U
        case 217 ... 220:
            out = 85;
            break;
        case 249 ... 252:
            out = 85;
            break;
        //C
        case 199:
            out = 67;
            break;
        case 231:
            out = 67;
            break;
        //OUTROS
        default:
            out = l;

    }
    //cout << "out: " << out << endl;
    return out;
}
```