Rapport de Projet
Indexation et recherche Multimédia

Yasser Rabi 2SIL1

I- Introduction :

1. Définition de l'indexation :

l'indexation automatique de documents est un domaine de l'informatique et des Sciences de l'information et des bibliothèques qui utilise des méthodes logicielles pour établir un index pour un ensemble de documents et faciliter l'accès ultérieur aux documents et à leur contenu. En revanche, les fichiers séquentiels indexés constituent une technique d'usage très général en informatique, pour le stockage de données numériques .

2. Technologie utilisé :

- C/C++
- QT

3. Terminologie :

- ### Mot vide :

En recherche d'information, les mots vides (ou stop words, en anglais) sont des mots qui sont tellement communs qu'il est inutile de les indexer ou de les utiliser dans une recherche. En français, des mots vides évidents pourraient être « le », « la », « de », « du », « ce », « ça »...

Un mot vide est un mot non significatif figurant dans un texte. On l'oppose à mot plein. La signification d'un mot s'évalue à partir de sa distribution (au sens statistique) dans une collection de textes. Un mot dont la distribution est uniforme sur les textes de la collection est dit « vide ». En d'autres termes, un mot qui apparaît avec une fréquence semblable dans chacun des textes de la collection n'est pas discriminant, ne permet pas de distinguer les textes les uns par rapport aux autres.

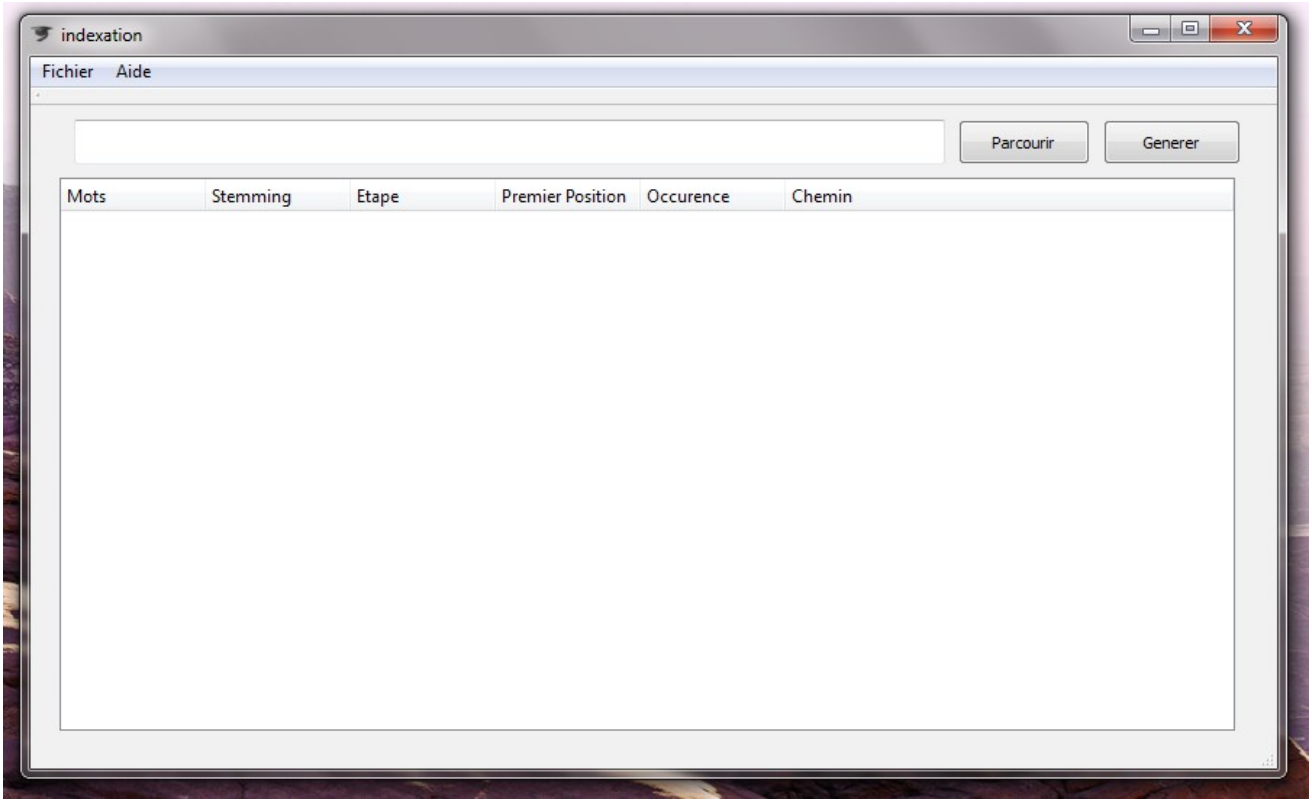
- ### Index ou Fichier inverse :

Un index est en toute généralité, une liste de descripteurs à chacun desquels est associée une liste des documents et/ou parties de documents auxquels ce descripteur renvoie. Ce renvoi peut être pondéré. Lors de la recherche d'information d'un usager, le système rapprochera la demande de l'index pour établir une liste de réponses. En amont, les méthodes utilisées pour constituer automatiquement un index pour un ensemble de documents varient considérablement avec la nature des contenus documentaires à indexer.

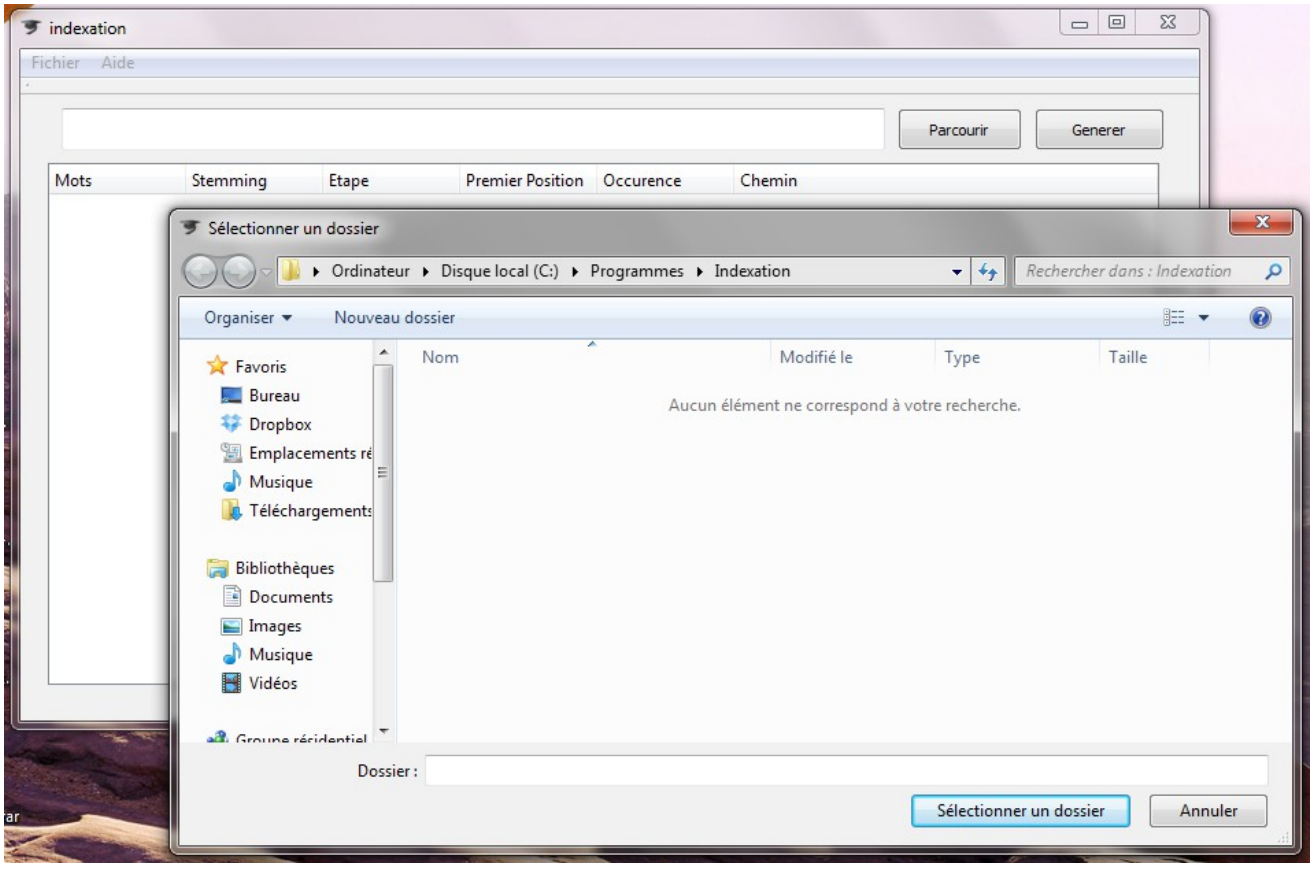
II- Travaille demandé :

- Extraction des mot d'un fichier texte et construire le fichier inversé contenant ces mot , leur récurrence et leur position ainsi qu'au fichier aux qu'elle appartient.
- Radicalisation des mots extrait et affichage des étape appliqué à chaque mot.

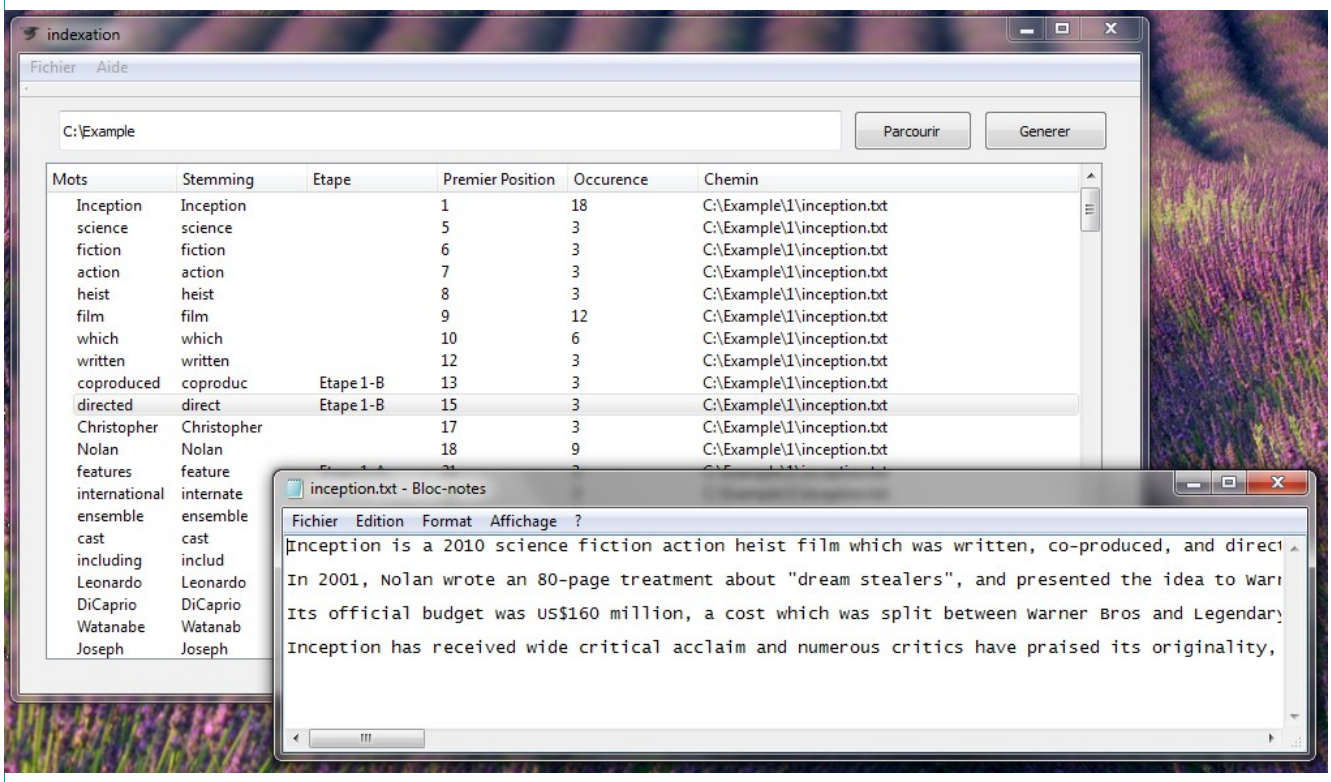
- Interface Graphique :



- Sélection de répertoire :



- Double Clic pour ouvrir le fichier texte correspondant :



- Classe de Base : « Index.h »

```
#include <stdio.h>
#include <iostream>
#include <sstream>
#include <dirent.h>
#include <string.h>
#include <algorithm>
#include <QString>

using namespace std;

// Déclaration d'une structure (Enregistrement) efin de bien organiser le
tableau d'Indexation.

typedef struct mot
{
    string mot;
    int pos;
    int rec;
    string chemin;
    string stem;
    string etape;
}en;
```

```
// Eneregistrement qui serve a manipuler les fichier texte trouvé dans une
repertoire donneé

typedef struct fich
{
    string nom;
    int nbr;
    string chemin;
}fichier;

// Class Principale du programme.

class findex
{
private:
    en * t;
    int taille;
    int p;
public:

// Constructeur de la classe findex .

findex(int x)
    {
        t=new en[x];
        p=0;
  }

// Verifier si un mot passé en paramètre existe dans l'index ou non.

bool verif(string nom)
    {
        for(int i=0;i<p;i++)
        {
            if (t[i].mot==nom)
            {
                t[i].rec++;
                return true;
            }
        }
        return false;
    }

// Ajouter un mot donné à l'indexe.

    void ajout(string mot,int pos,string chemin)
    {
        if(!verif(mot))
        {
```

```
            t[p].mot=mot;
            t[p].pos=pos;
            t[p].chemin=chemin;
            t[p].rec=1;
            t[p].stem=steming(mot,p);
            p++;
        }
    }

// Méthode permettant de récupérer le nombre de mot stocker dans l'indexe.

    int getp()
    {
        return p;
    }

// Sauvgarder le fichier inverse.

    void save()
    {
        cout<<"save appele";
        QString CurrentDir = QDir::currentPath()+"\\Index.txt";
        FILE * resultat=fopen((char*)CurrentDir.toStdString().c_str(),"w+");
        string var;
        for(int i=0;i<p;i++)
                {
                    stringstream tmp;
                    tmp << t[i].pos;
                    stringstream tmp2;
                    tmp2 << t[i].rec;
                    var=t[i].mot+";"+tmp.str()+";"+tmp2.str()+"\n";
                    fputs((char*)var.c_str(),resultat);
                }
        fclose(resultat);
    }

// Méthode Permettant de récupérer le tableau d'index.

    en* gett()
    {
        return t;
    }

// Méthode Permettant de transformer une chaîne donnée en majiscule .

    string upper(string strToConvert)
    {
        transform(strToConvert.begin(), strToConvert.end(),
        strToConvert.begin(), ::toupper);

        return strToConvert;
    }
```

```
// Méthode Permettant de vérifier terminaison du mot .

bool term(string mot,string term)
    {
        int x=term.length();
        if (x<mot.length())
        return (upper(mot.substr((mot.length()-x),term.length()))==term);
        else
            return false;
    }

// Méthode Permettant de remplacer la terminaison par une autre.

string remplace(string mot,string term1,string term2)
    {
        int x1=term1.length();
        return mot.replace(mot.length()-x1,x1,term2);
   }

// Compter le nombre VC dans un mot donnée .

 int m(string mot)
    {
        char* ch=(char*)mot.c_str();
        string voyelle="aeyuio";
        int v=0,m=0;
        for(int i=0;i<(mot.length()-1);i++)
        {
            if(voyelle.find(ch[i]))
                v++;
            else
            {
                if(v)
                {
                    v=0;
                    m++;
                }
            }
        }
        return m;
    }

// Verifier si un mot reste valable apres le remplacement par une terminaison
donneé.


    bool v(string mot,string term1,string term2)
    {
        char* ch=(char*) remplace(mot,term1,term2).c_str();
```

```
        string voyelle="aeyuio";
        int v=0,c=0;
        for(int i=0;i<(mot.length()-1);i++)
        {
            if(!voyelle.find(ch[i]))
            {
                c++;
                if(v)
                {
                    return true;
                }
            }
            else
            {
                if(v)
                {
                    return true;
                }
                if(c)
                {
                    c=0;
                    v++;
                }
            }
        }
    }

// Etape 1a de l'algorithme de porter.

    string e1a(string mot)
    {
        if(term(mot,"SSES"))
            mot=remplace(mot,"sses","ss");
        else if(term(mot,"IES"))
            mot=remplace(mot,"ies","i");
        else if(term(mot,"SS"))
            ;
        else if(term(mot,"S"))
            mot=remplace(mot,"S","");
        return mot;
    }

// Etape 1b de l'algorithme de porter.

    string e1b(string mot)
    {
        if((term(mot,"EED"))&&(m(mot)>0))
            mot=remplace(mot,"eed","ee");
        else if ((term(mot,"ED"))&&(v(mot,"ed","")))
            mot=remplace(mot,"ed","");
        else if ((term(mot,"ING"))&&(v(mot,"ing","")))
            mot=remplace(mot,"ing","");
        return mot;
```

```
    }


// Etape 1c de l'algorithme de porter.

string e1c(string mot)
    {
        if((term(mot,"Y"))&&(v(mot,"y","i")))
            remplace(mot,"y","i");
        return mot;
    }

// Etape 2 de l'algorithme de porter.

string e2(string mot)
    {
        if((term(mot,"ATIONAL"))&&(m(mot)>0))
            mot=remplace(mot,"ational","ate");
        else if((term(mot,"TIONAL"))&&(m(mot)>0))
            mot=remplace(mot,"tional","tion");
        else if((term(mot,"ENCI"))&&(m(mot)>0))
            mot=remplace(mot,"enci","ence");
        else if((term(mot,"ANCI"))&&(m(mot)>0))
            mot=remplace(mot,"anci","ance");
        else if((term(mot,"IZER"))&&(m(mot)>0))
            mot=remplace(mot,"izer","ize");
        else if((term(mot,"ABLI"))&&(m(mot)>0))
            mot=remplace(mot,"abli","able");
        else if((term(mot,"ALLI"))&&(m(mot)>0))
            mot=remplace(mot,"alli","al");
        else if((term(mot,"ENTLI"))&&(m(mot)>0))
            mot=remplace(mot,"entli","ent");
        else if((term(mot,"ELI"))&&(m(mot)>0))
            mot=remplace(mot,"eli","E");
        else if((term(mot,"OUSLI"))&&(m(mot)>0))
            mot=remplace(mot,"ousli","ous");
        return mot;
    }

// Etape 3 de l'algorithme de porter.

string e3(string mot)
    {
        if((term(mot,"ICATE"))&&(m(mot)>0))
            mot=remplace(mot,"icate","ic");
        else if((term(mot,"ATIVE"))&&(m(mot)>0))
            mot=remplace(mot,"ative","");
        else if((term(mot,"ALIZE"))&&(m(mot)>0))
            mot=remplace(mot,"alize","al");
        else if((term(mot,"ICITI"))&&(m(mot)>0))
            mot=remplace(mot,"iciti","ic");
        else if((term(mot,"ICAL"))&&(m(mot)>0))
```

```
            mot=remplace(mot,"ical","ic");
        else if((term(mot,"FUL"))&&(m(mot)>0))
            mot=remplace(mot,"full","");
        else if((term(mot,"NESS"))&&(m(mot)>0))
            mot=remplace(mot,"ness","");
        return mot;
    }

 // Etape 4 de l'algorithme de porter.

string e4(string mot)
    {
        if((term(mot,"AL"))&&(m(mot)>1))
            mot=remplace(mot,"ational","");
        else if((term(mot,"TIONAL"))&&(m(mot)>1))
            mot=remplace(mot,"ance","");
        else if((term(mot,"ENCE"))&&(m(mot)>1))
            mot=remplace(mot,"ence","");
        else if((term(mot,"ER"))&&(m(mot)>1))
            mot=remplace(mot,"er","");
        else if((term(mot,"IC"))&&(m(mot)>1))
            mot=remplace(mot,"ic","");
        else if((term(mot,"ABLE"))&&(m(mot)>1))
            mot=remplace(mot,"able","");
        return mot;
   }

// Etape 5 de l'algorithme de porter.

string e5(string mot)
    {
        if((term(mot,"E"))&&(m(mot)>1))
            mot=remplace(mot,"e","");
        else if((term(mot,"E"))&&(m(mot)==1))
           ;
        else if((term(mot,"E"))&&(m(mot)>1))
             ;
        return mot;
    }

// Méthode faisant appel a toute les étape de l'algorithme de porter.

string steming(string mot,int pos)

    {
        if(e1a(mot)!=mot)
                {
                    mot=e1a(mot);
                    t[p].etape+="  Etape 1-A";
                }
                if(e1b(mot)!=mot)
                {
```

```
                mot=e1b(mot);
                t[p].etape+="  Etape 1-B ";
            }
            if(e1c(mot)!=mot)
            {
                mot=e1c(mot);
                t[p].etape+="  Etape 1-C";
            }
            if(e2(mot)!=mot)
            {
                mot=e2(mot);
                t[p].etape+="  Etape 2";
            }
            if(e3(mot)!=mot)
            {
                mot=e3(mot);
                t[p].etape+="  Etape 3";
            }
            if(e4(mot)!=mot)
            {
                mot=e4(mot);
                t[p].etape+="  Etape 4";
            }
            if(e5(mot)!=mot)
            {
                mot=e5(mot);
                t[p].etape+="  Etape 5";
            }
        return mot;
    }

// Méthode qui index un fichier texte passé en paramètre  .

    void index(string nom,string chemin)
    {
        char c=0;
        int i=0,j=0;
        string var;
        int pos=0;
        size_t found=0;
        string exep="/*-+.²&\"'(-_=?§µ$£^¨<>+°;:,[]1234567890";
        FILE * index=fopen((chemin+nom).c_str(),"r");
        if (index==NULL)
                cout<<"\n ERREUR, Ce Fichier n'existe pas\n\n";
        else
            {
                while(c!=EOF)
                {
                    c=getc(index);
                    i++;
                    if (c!=EOF)
                    {
                        if ((c!= ' ')&&(c!='\n'))
```

```
                                {
                                found=exep.find(c);
                                    if(found==string::npos)
                                    {
                                        var+=c;
                                        //j++;
                                    }
                                }
                            else
                            {
                                j++;
                                if(var.length()>3)
                                {
                                    ajout(var,j,chemin+nom);
                                }
                                var="";
                                //j=0;
                            }
                        }
                    }
                    fclose(index);
                }
    }
};
```

III- Référence :

- http://qt.nokia.com/
- http://qt.developpez.com/
- http://cpp.developpez.com/