TP02 : boîte de dialogue pour trouver des fichiers dans un dossier spécifié

Objectif :

L'application Rechercher des fichiers permet à l'utilisateur de rechercher des fichiers dans un répertoire spécifié, correspondant à un nom de fichier donné ou à un caractère générique, et contenant une chaîne spécifiée. Le résultat de la recherche est affiché dans un tableau contenant les noms des fichiers et leur taille. L'application indique également le nombre de fichiers trouvés.

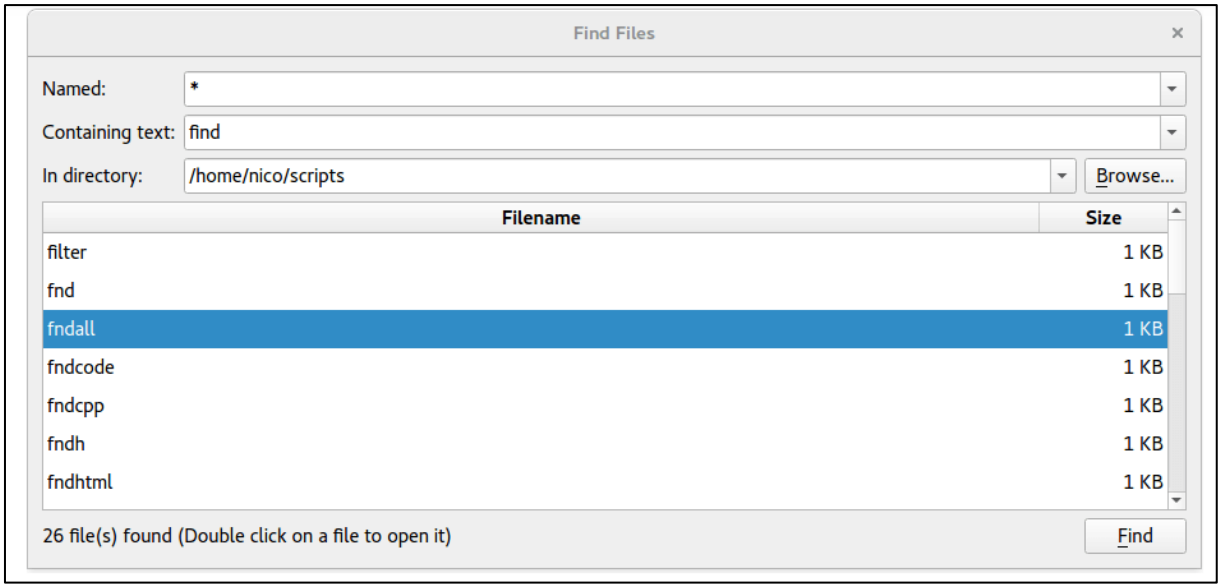
Travail demandé :

- Suivre les étapes du TP pour construire l'application.
- Documentation de l'application

Pour mettre en œuvre cette application plusieurs classes doivent être utilisées. Le tableau ci-dessous montre ces classes :

QProgressDialog	Fournir des informations sur l'avancement d'une opération de recherche
QFileDialog	Parcourir une liste de fichiers
QTextStream	Utiliser des opérateurs de flux pour lire un fichier
QTableWidget	Parcourir les résultats de la recherche dans un tableau
QDesktopServices	Ouvrir des fichiers dans la liste des résultats dans une application appropriée

La figure 1 représente le visuel de l'application finale :



-Figure 1 : boîte de dialogue pour trouver des fichiers dans un dossier spécifié-

Etape 1: Définition de classe de fenêtre (Window.h)

La classe Window hérite de QWidget et constitue le widget d'application principal. Il affiche les options de recherche et affiche les résultats de la recherche.

```
class Window : public QWidget
{
    Q_OBJECT

public:
    Window(QWidget *parent = 0);

private slots:
    void browse();
    void find();
    void animateFindClick();
    void openFileOfItem(int row, int column);
    void contextMenu(const QPoint &pos);

private:
    QStringList findFiles(const QStringList &files, const QString &text);
    void showFiles(const QStringList &paths);
```

```
    QComboBox *createComboBox(const QString &text = QString());
    void createFilesTable();

    QComboBox *fileComboBox;
    QComboBox *textComboBox;
    QComboBox *directoryComboBox;
    QLabel *filesFoundLabel;
    QPushButton *findButton;
    QTableWidget *filesTable;

    QDir currentDir;
};
```

L'application dispose de deux slots privés:

- browse() : appelé chaque fois que l'utilisateur souhaite parcourir un répertoire dans lequel il veut effectuer une recherche.
- find () : à chaque fois l'utilisateur lance une recherche avec le bouton Rechercher ce slot sera appelé.

De plus, nous déclarons d'autres fonctions privées:

- findFiles () : elle recherche les fichiers correspondant aux paramètres de recherche.
- showFiles () Afficher le résultat de la recherche.
- createComboBox () : elle construit le widget.
- createFilesTable() : elle construit le widget.

Etape 2: Implémentation de la classe (Window.cpp)

0- Dans le constructeur, nous créons d'abord les widgets de l'application.

```
Window::Window(QWidget *parent)
    : QWidget(parent)
{
    setWindowTitle(tr("Find Files"));
    QPushButton *browseButton = new QPushButton(tr("&Browse..."), this);
    connect(browseButton, &QAbstractButton::clicked, this, &Window::browse);
    findButton = new QPushButton(tr("&Find"), this);
    connect(findButton, &QAbstractButton::clicked, this, &Window::find);

    fileComboBox = createComboBox(tr("*"));
    connect(fileComboBox->lineEdit(), &QLineEdit::returnPressed,
            this, &Window::animateFindClick);
    textComboBox = createComboBox();
    connect(textComboBox->lineEdit(), &QLineEdit::returnPressed,
            this, &Window::animateFindClick);
    directoryComboBox = createComboBox(QDir::toNativeSeparators(QDir::currentPath()));
    connect(directoryComboBox->lineEdit(), &QLineEdit::returnPressed,
            this, &Window::animateFindClick);

    filesFoundLabel = new QLabel;

    createFilesTable();

    QGridLayout *mainLayout = new QGridLayout(this);
    mainLayout->addWidget(new QLabel(tr("Named:")), 0, 0);
```

```
    mainLayout->addWidget(fileComboBox, 0, 1, 1, 2);
    mainLayout->addWidget(new QLabel(tr("Containing text:")), 1, 0);
    mainLayout->addWidget(textComboBox, 1, 1, 1, 2);
    mainLayout->addWidget(new QLabel(tr("In directory:")), 2, 0);
    mainLayout->addWidget(directoryComboBox, 2, 1);
    mainLayout->addWidget(browseButton, 2, 2);
    mainLayout->addWidget(filesTable, 3, 0, 1, 3);
    mainLayout->addWidget(filesFoundLabel, 4, 0, 1, 2);
    mainLayout->addWidget(findButton, 4, 2);
```

Nous créons les widgets pour construire l'interface utilisateur et nous les ajoutons à une layout principale à l'aide de QGridLayout.

1- Nous n'avons pas créé de QMenuBar avec un élément de menu Quitter; mais nous aimerions toujours avoir un raccourci clavier pour quitter. Pour cela nous construisons un QShortcut avec QKeySequence::Quit et que nous le connectons à QApplication::quit (), sur la plupart des plates-formes, il sera possible d'appuyer sur Control-Q pour quitter (ou selon la clé de sortie standard configurée sur cette plate-forme). Sur macOS, cela est redondant, car chaque application obtient automatiquement un élément de menu Quitter, mais cela permet de rendre l'application portable.

```
connect(new QShortcut(QKeySequence::Quit, this), &QShortcut::activated,
        qApp, &QApplication::quit);
```

2- Le slot browse () présente une boîte de dialogue de fichier pour l'utilisateur, à l'aide de la classe QFileDialog. QFileDialog permet à un utilisateur de parcourir le système de fichiers afin de sélectionner un ou plusieurs fichiers ou un répertoire. Le moyen le plus simple de créer un fichier QFileDialog consiste à utiliser les fonctions statiques pratiques.
Nous utilisons ici la fonction statique QFileDialog::getExistingDirectory() qui renvoie un répertoire existant sélectionné par l'utilisateur. Ensuite, nous affichons le répertoire dans la liste déroulante de répertoire à l'aide de la fonction QComboBox::addItem() et mettons à jour les indices actuels.
QComboBox::addItem() ajoute un élément à la liste déroulante avec le texte donné (s'il ne figure pas déjà dans la liste) et contenant le userData spécifié. L'élément est ajouté à la liste des éléments existants.

```
void Window::browse()
{
    QString directory =
        QDir::toNativeSeparators(QFileDialog::getExistingDirectory(this, tr("Find Files"),
QDir::currentPath()));

    if (!directory.isEmpty()) {
        if (directoryComboBox->findText(directory) == -1)
            directoryComboBox->addItem(directory);
        directoryComboBox->setCurrentIndex(directoryComboBox->findText(directory));
    }
}
```

3- Le slot find () est appelé chaque fois que l'utilisateur demande une nouvelle recherche en appuyant sur le bouton Find.
Tout d'abord, nous éliminons tous les résultats de recherche précédents en remettant le nombre de lignes du widget de table à zéro. Ensuite, nous récupérons le nom de fichier spécifié, le texte et le chemin du répertoire à partir des listes déroulantes respectives.
Nous utilisons le chemin du répertoire pour créer un QDir; la classe QDir fournit un accès à la structure de répertoires et à son contenu.

```
void Window::find()
{
    filesTable->setRowCount(0);

    QString fileName = fileComboBox->currentText();
    QString text = textComboBox->currentText();
    QString path = QDir::cleanPath(directoryComboBox->currentText());
    currentDir = QDir(path);
```

Nous utilisons QDirIterator pour parcourir les fichiers qui correspondent au nom de fichier spécifié et créer une liste QStringList de chemins.

Ensuite, nous recherchons dans tous les fichiers de la liste, en utilisant la fonction privée findFiles(), en éliminant ceux qui ne contiennent pas le texte spécifié. Nous les trions (car QDirIterator ne l'avait pas). Enfin, nous affichons les résultats à l'aide de la fonction privée showFiles ().

Si l'utilisateur n'a pas spécifié de texte, il n'y a aucune raison de rechercher dans les fichiers. Nous trions et affichons les résultats immédiatement.

```
QStringList filter;
    if (!fileName.isEmpty())
        filter << fileName;
    QDirIterator it(path, filter, QDir::AllEntries | QDir::NoSymLinks | QDir::NoDotAndDotDot,
QDirIterator::Subdirectories);
    QStringList files;
    while (it.hasNext())
        files << it.next();
    if (!text.isEmpty())
        files = findFiles(files, text);
    files.sort();
    showFiles(files);
}
```