

Computer Vision

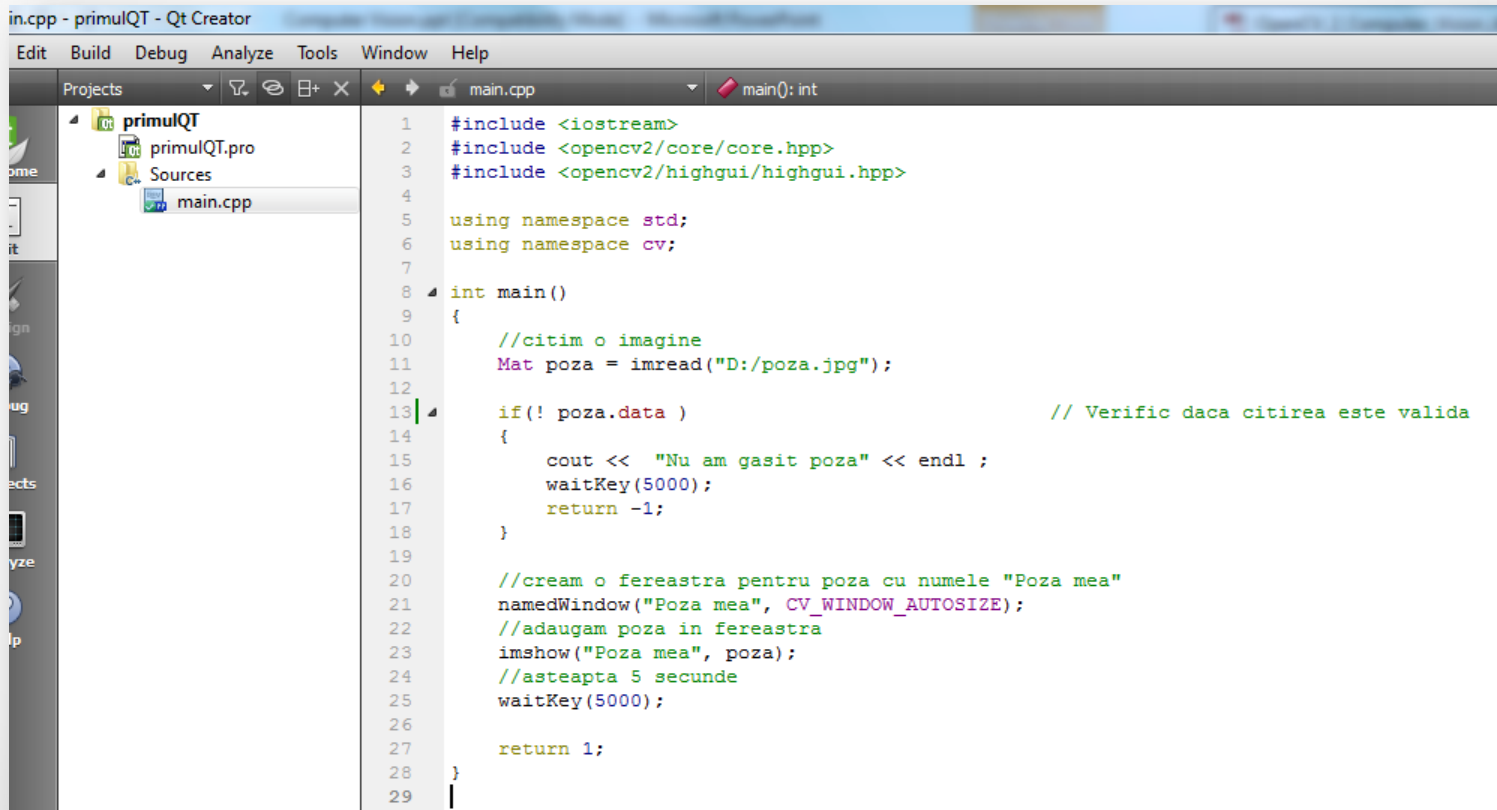
Catalin Stoean
catalin.stoean@inf.ucv.ro
<http://inf.ucv.ro/~cstoean>

Obiective

- Cele mai uzuale module din OpenCV
- Sa intelegem cateva din obiectele si metodele cele mai utilizate in OpenCV
- Oglindirea unei imagini
- Rularea unui fisier video

Proiect OpenCV folosind QT

- Ne amintim aplicatia de data trecuta.



```
1 #include <iostream>
2 #include <opencv2/core/core.hpp>
3 #include <opencv2/highgui/highgui.hpp>
4
5 using namespace std;
6 using namespace cv;
7
8 int main()
9 {
10     //citim o imagine
11     Mat poza = imread("D:/poza.jpg");
12
13     if(! poza.data ) // Verific daca citirea este valida
14     {
15         cout << "Nu am gasit poza" << endl ;
16         waitKey(5000);
17         return -1;
18     }
19
20     //cream o fereastră pentru poza cu numele "Poza mea"
21     namedWindow("Poza mea", CV_WINDOW_AUTOSIZE);
22     //adaugam poza in fereastră
23     imshow("Poza mea", poza);
24     //asteapta 5 secunde
25     waitKey(5000);
26
27     return 1;
28 }
29
```

Module OpenCV

- **opencv_core** – contine functionalitati de baza ale libreriei, structuri de baza si functii aritmetice
- **opencv_imgproc** – contine principalele functii de procesare ale imaginilor
- **opencv_highgui** – contine functii pentru citirea si scrierea fisierelor cu imagini si video si alte functii pentru interfata
- **opencv_features2d** – contine functii pentru detectarea de puncte de interes

Module OpenCV

- **opencv_calib3d** – contine functii pentru calibrarea camerei, estimare geometrica pentru doua perspective, functii stereo
- **opencv_video** – contine functii despre estimarea miscarii, urmarire de caracteristici, extragere de informatii din prim plan.
- **opencv_objdetect** – contine functii pentru detectare de obiecte (de exemplu, a feței)

Stocarea unei imagini in OpenCV

- Obiectele din cadrul OpenCV utilizate pana acum sunt de tip `cv`.
 - Acesta este motivul pentru care am adaugat **using namespace cv**;
- Stocarea unei imagini se face intr-un obiect de tip **Mat**.
 - `Mat poza; //daca avem in cadrul programului using namespace cv`
 - `cv::Mat poza; //altfel`
- Definirea creeaza o imagine de marime 0 x 0.
- Pentru a vedea dimensiunea imaginii, putem metoda **size()** a lui `Mat`:

```
cout<<"Inaltimea este de "<<poza.size().height<<
      ", iar lungimea de "<<poza.size().width<<".<<endl;
```

- De adaugat aceasta bucata de cod la programul cu afisarea pozei. ●

Citirea unei imagini

- Se utilizeaza metoda **imread** cu argumentul dat de un string ce contine calea catre fisierul cu poza.
 - `Poza = imread("D:/yes.jpg");`
 - Poza este citita din fisier, decodata si i se aloca memorie
- Pentru a verifica daca a fost buna calea sau formatul este acceptat
 - `if(! poza.data)`
- Pointerul **data** este 0 atunci cand nu s-a citit imaginea.

Afisarea imaginii

- Este utilizat modulul `highgui` al OpenCV pentru a afisa imaginile.
- `namedWindow("poza mea", CV_WINDOW_AUTOSIZE);`
- `imshow("poza mea", poza);`
- **namedWindow** declara fereastra in care se doreste afisarea pozei
- **imshow** specifica faptul ca poza ar trebui sa apara in acea fereastra.

Mat si IplImage

- Odata cu OpenCV 2, o noua interfata C++ a fost introdusa.
- In interfata anterioara, imaginile erau reprezentate folosind structura IplImage.
- Se poate face insa convertire intre IplImage si Mat:
- ```
IplImage* iplImage =
cvLoadImage("c:\\img.jpg");
Mat poza(iplImage);
```
- Pentru multitudinea de constructori Mat vedeti:  
[http://docs.opencv.org/modules/core/doc/basic\\_structures.html#mat-mat](http://docs.opencv.org/modules/core/doc/basic_structures.html#mat-mat)

# Oglindirea unei imagini

- Pentru a nu modifica poza initiala, se declara o noua variabila de tip **Mat**.
- `Mat rezultat;`
- Se apeleaza apoi metoda `flip`
- `flip(poza, rezultat, 1)`
- Apoi se afiseaza cea de a doua imagine:
- `namedWindow("poza oglindita", CV_WINDOW_AUTOSIZE);`
- `imshow("poza oglindita", rezultat);`
- `waitKey(0); //pentru a astepta atingerea //unei taste`

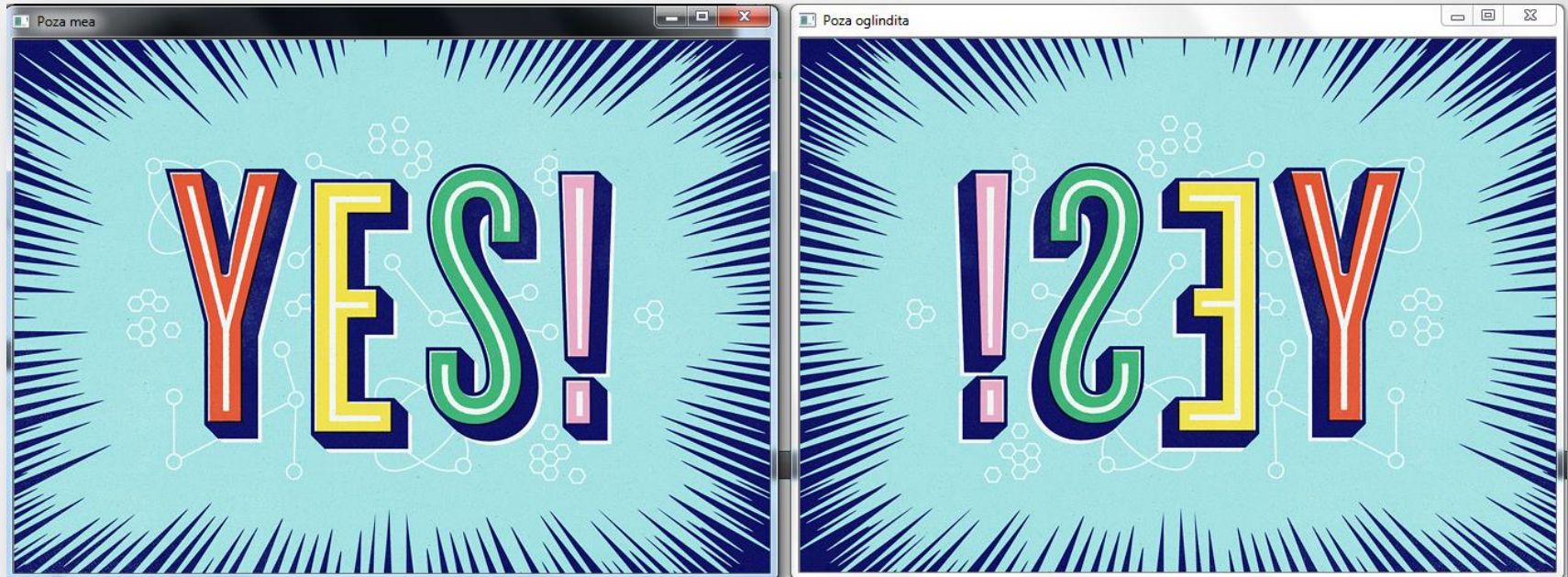
# Oglindirea unei imagini

- Cel de-l treilea argument de la metoda **flip** poate fi
  - Pozitiv: oglindire orizontala
  - Zero: oglindire verticala
  - Negativ: si orizontala, si verticala
- **waitKey** este tot o metoda a modulului highgui.
- Imaginea poate fi salvata folosind metoda **imwrite**.

# Oglindirea unei imagini

- `imwrite("D:\oglundita.jpg", rezultat);`
- Primul argument este calea catre fisier si numele, cel de-al doilea variabila de tip `Mat` care contine poza de salvat.
- Poza poate fi salvata si in alte formate: `bmp`, `png` etc. si in functie de extensie se stabileste codecul folosit.

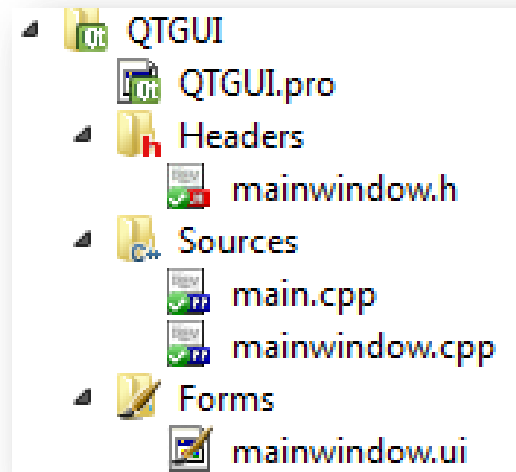
# Rezultatul



- Modificati programul pentru a afisa 4 poze concomitent: originala, oglindita in sus, in jos si in ambele directii.
  - Definiti o metoda pe care o apelati de 4 ori pentru a face afisarea unei imagini.

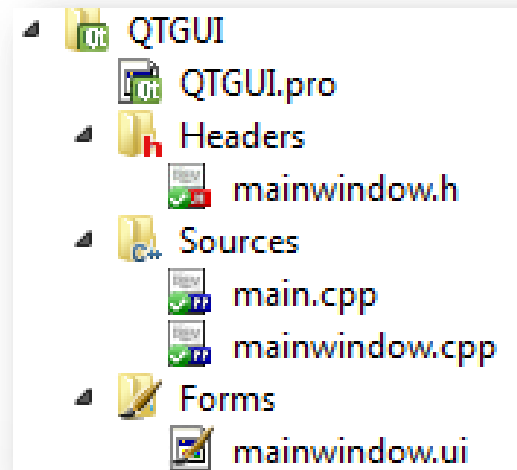
# Interfata in Qt pentru oglindirea unei imagini

- Cream un proiect in Qt de tip **QT Widgets Application**.
- In cadrul acestuia se vor crea urmatoarele fisiere:



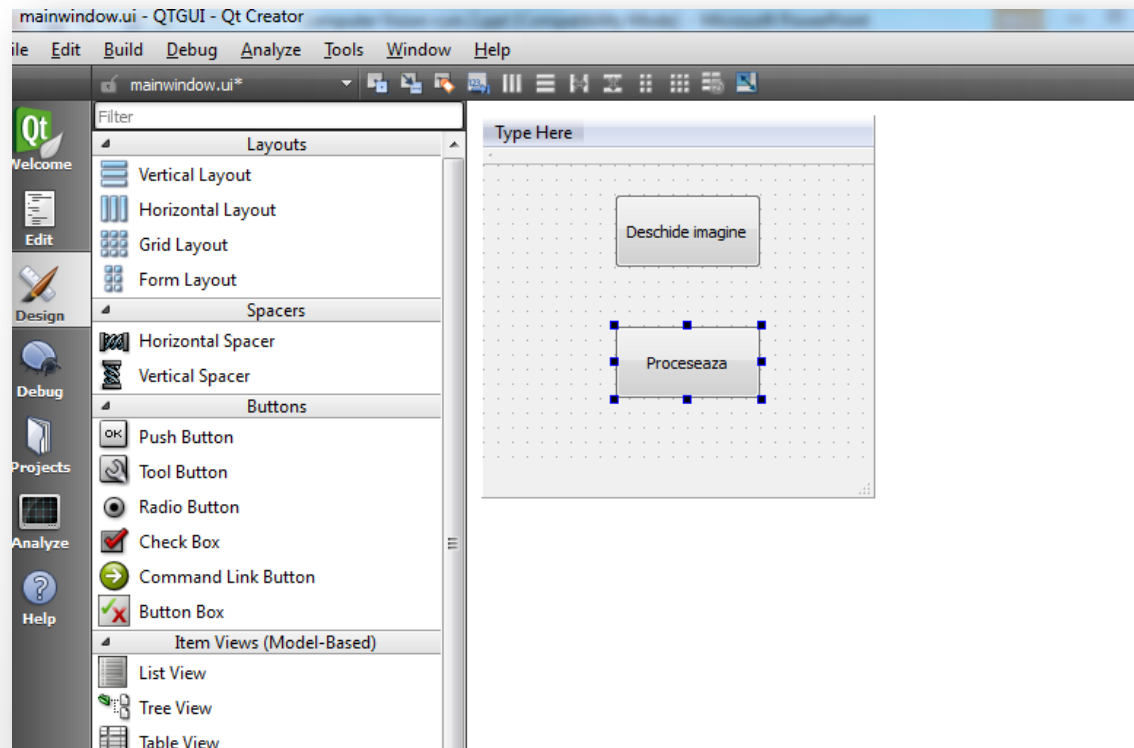
# Interfata in Qt pentru oglindirea unei imagini

- In fisierul proiectului (cel cu extensia **.pro**) nu uitam sa adaugam librariile OpenCV.
- Fisierul **mainwindow.cpp** defineste clasa care contine componentele ferestrei GUI.
- Fisierul **mainwindow.ui** descrie aspectul ferestrei GUI.
- Fisierul **mainwindow.h** este fisierul header al clasei mainwindow.



# Interfata in Qt pentru oglindirea unei imagini

- Dam dublu-click pe fisierul **mainwindow.ui**.
- Adaugam doua butoane carora le editam etichetele.

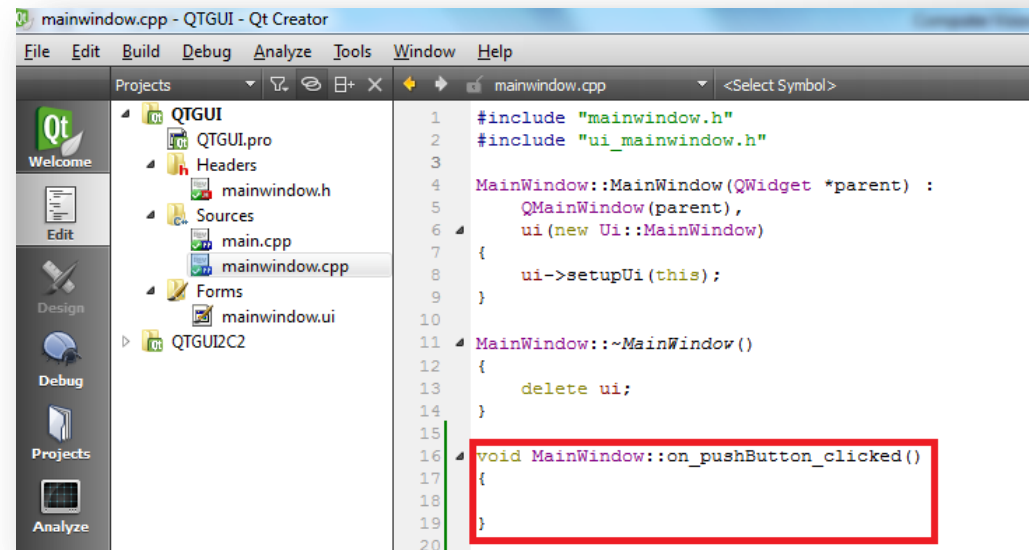




# Interfata in Qt pentru oglindirea unei imagini

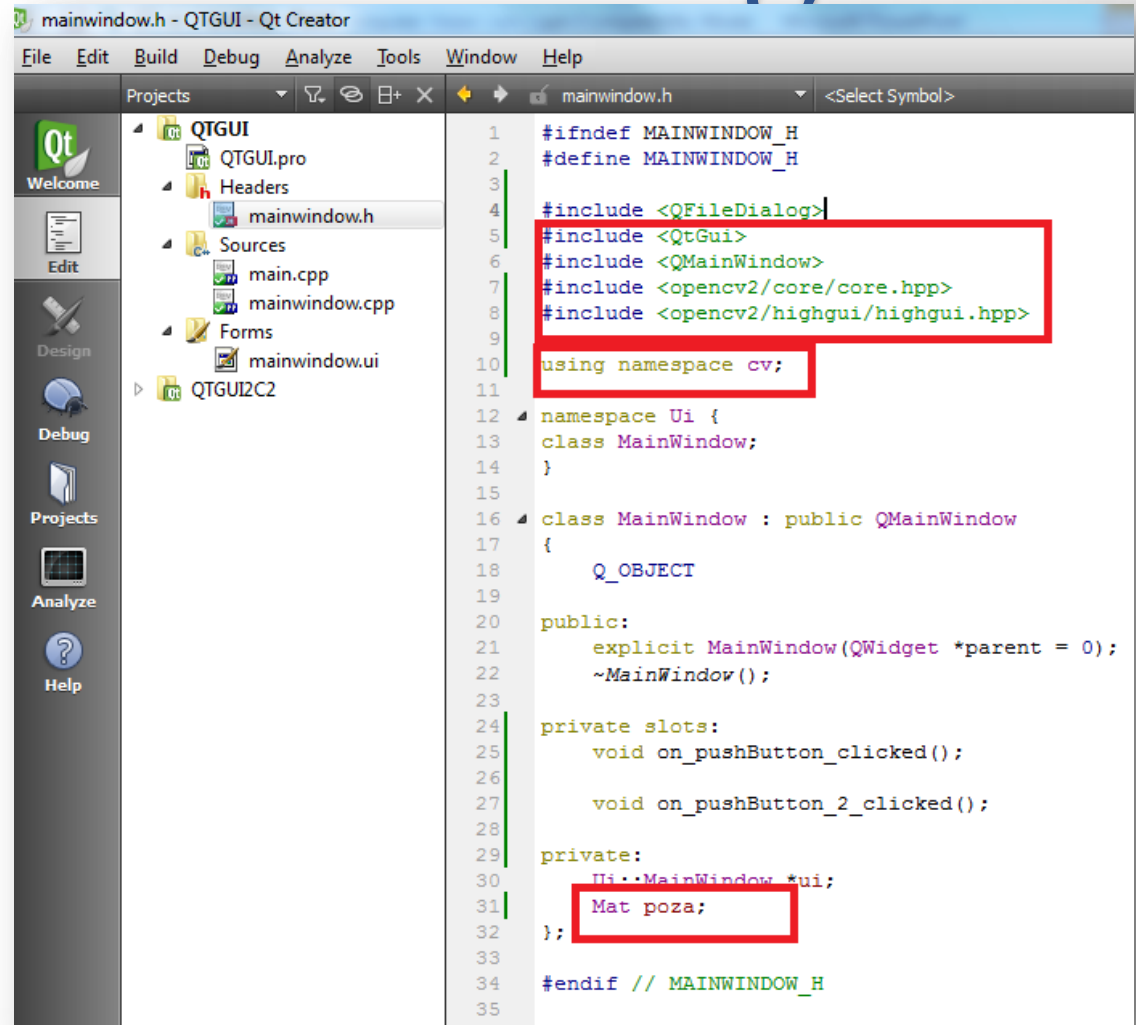
- Click-dreapta pe primul buton -> **Go to slot...** -> **clicked()**.
- Acesta face sa apara in mainwindow.cpp metoda **on\_pushButton\_clicked()**.

- Procedam apoi la fel si pentru al doilea buton.



```
mainwindow.cpp - QTGUI - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects mainwindow.cpp <Select Symbol>
Qt Welcome
Edit
Design
Debug
Projects
Analyze
QTGUI
 QTGUI.pro
 Headers
 mainwindow.h
 Sources
 main.cpp
 mainwindow.cpp
 Forms
 mainwindow.ui
 QTGUI2C2
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 MainWindow::MainWindow(QWidget *parent) :
5 QMainWindow(parent),
6 ui(new Ui::MainWindow)
7 {
8 ui->setupUi(this);
9 }
10
11 MainWindow::~MainWindow()
12 {
13 delete ui;
14 }
15
16 void MainWindow::on_pushButton_clicked()
17 {
18 }
19
20
```

# Interfata in Qt pentru oglindirea unei imagini

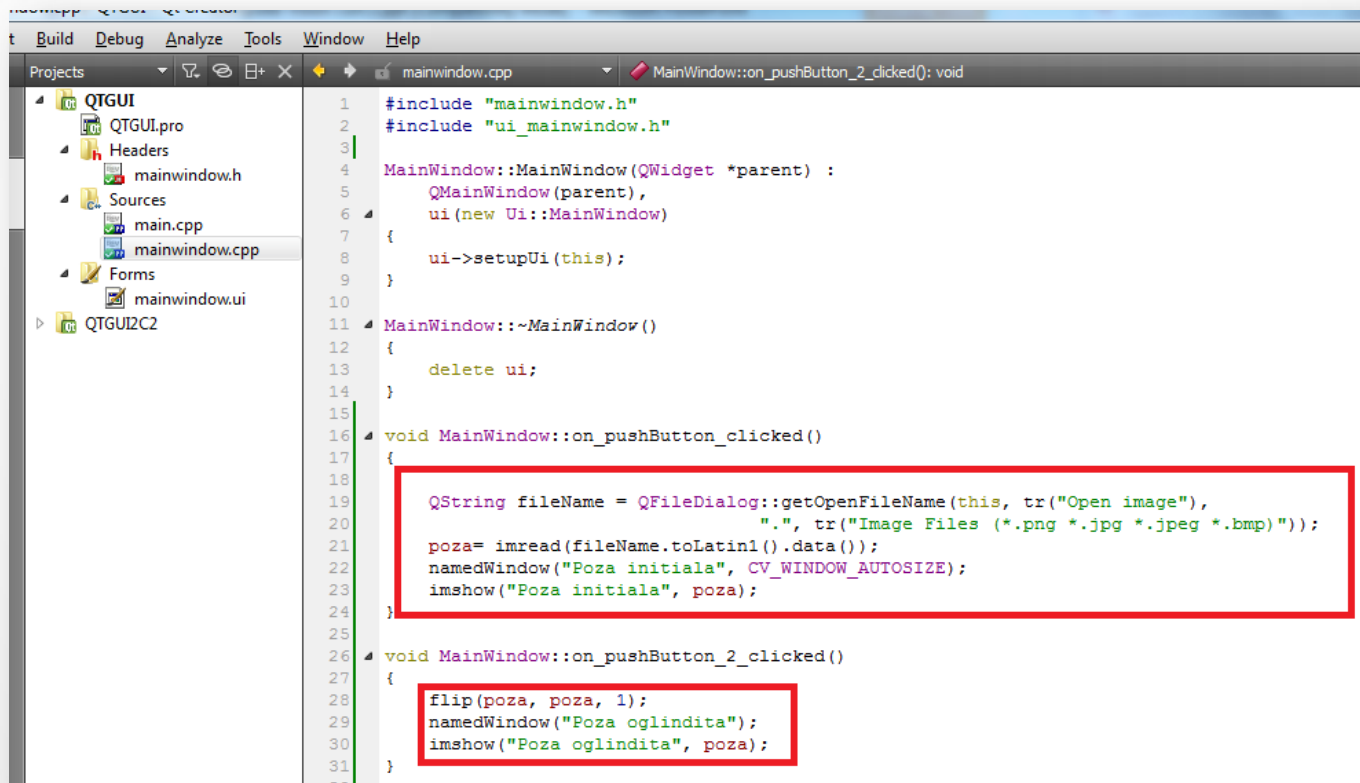


```
mainwindow.h - QTGUI - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects mainwindow.h <Select Symbol>
Qt
Welcome
Edit
Design
Debug
Projects
Analyze
Help
QTGUI
 QTGUI.pro
 Headers
 mainwindow.h
 Sources
 main.cpp
 mainwindow.cpp
 Forms
 mainwindow.ui
 QTGUI2C2
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include <QFileDialog>
5 #include <QtGui>
6 #include <QMainWindow>
7 #include <opencv2/core/core.hpp>
8 #include <opencv2/highgui/highgui.hpp>
9
10 using namespace cv;
11
12 namespace Ui {
13 class MainWindow;
14 }
15
16 class MainWindow : public QMainWindow
17 {
18 Q_OBJECT
19
20 public:
21 explicit MainWindow(QWidget *parent = 0);
22 ~MainWindow();
23
24 private slots:
25 void on_pushButton_clicked();
26
27 void on_pushButton_2_clicked();
28
29 private:
30 Ui::MainWindow *ui;
31 Mat poza;
32 };
33
34 #endif // MAINWINDOW_H
35
```

- In fisierul header, adaugam o variabila de tip Mat si includem modulele care ne sunt necesare.

# Interfata in Qt pentru oglindirea unei imagini

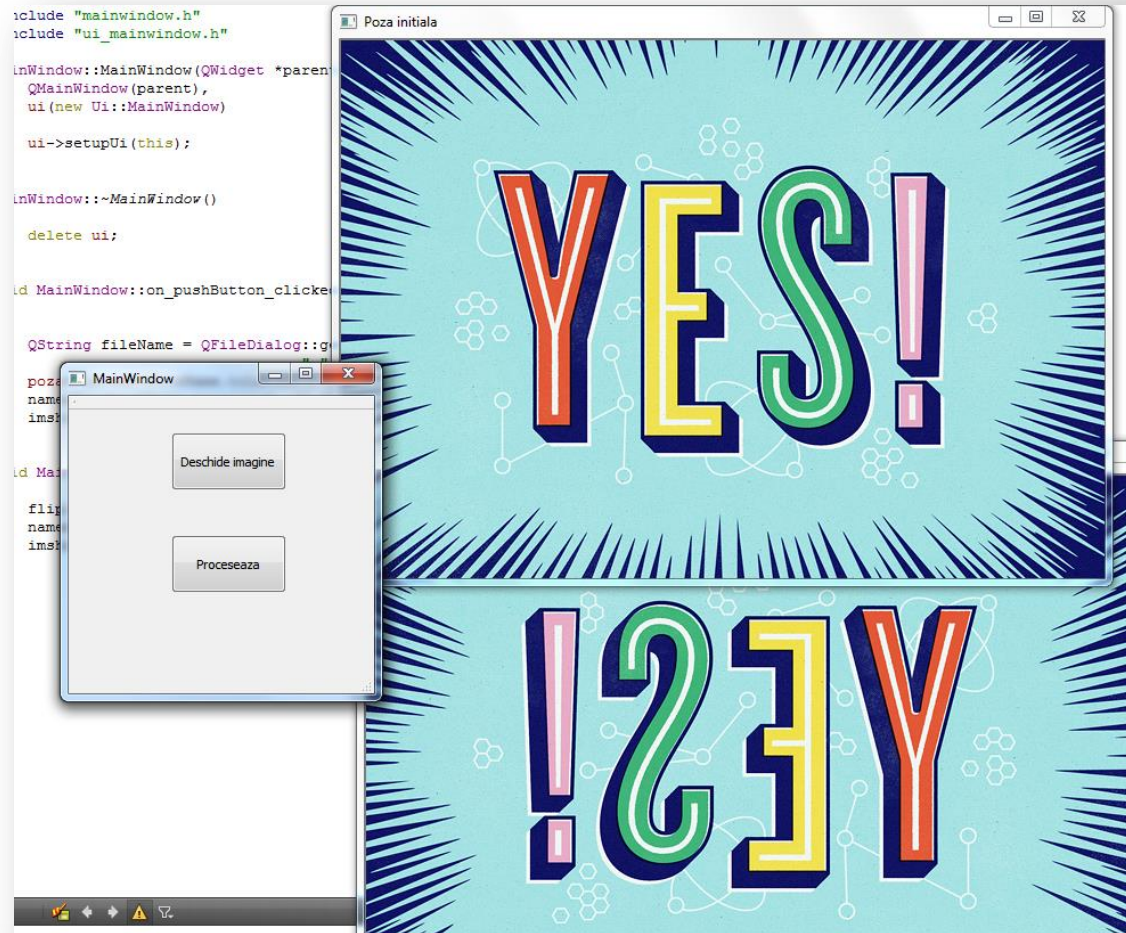
- In mainwindow.cpp adaugam ce trebuie sa faca aplicatia atunci cand sunt apasate butoanele.



```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 MainWindow::MainWindow(QWidget *parent) :
5 QMainWindow(parent),
6 ui(new Ui::MainWindow)
7 {
8 ui->setupUi(this);
9 }
10
11 MainWindow::~MainWindow()
12 {
13 delete ui;
14 }
15
16 void MainWindow::on_pushButton_clicked()
17 {
18 QString fileName = QFileDialog::getOpenFileName(this, tr("Open image"),
19 ".", tr("Image Files (*.png *.jpg *.jpeg *.bmp)"));
20 poza= imread(fileName.toLatin1().data());
21 namedWindow("Poza initiala", CV_WINDOW_AUTOSIZE);
22 imshow("Poza initiala", poza);
23 }
24
25
26 void MainWindow::on_pushButton_2_clicked()
27 {
28 flip(poza, poza, 1);
29 namedWindow("Poza oglindita");
30 imshow("Poza oglindita", poza);
31 }
32
```

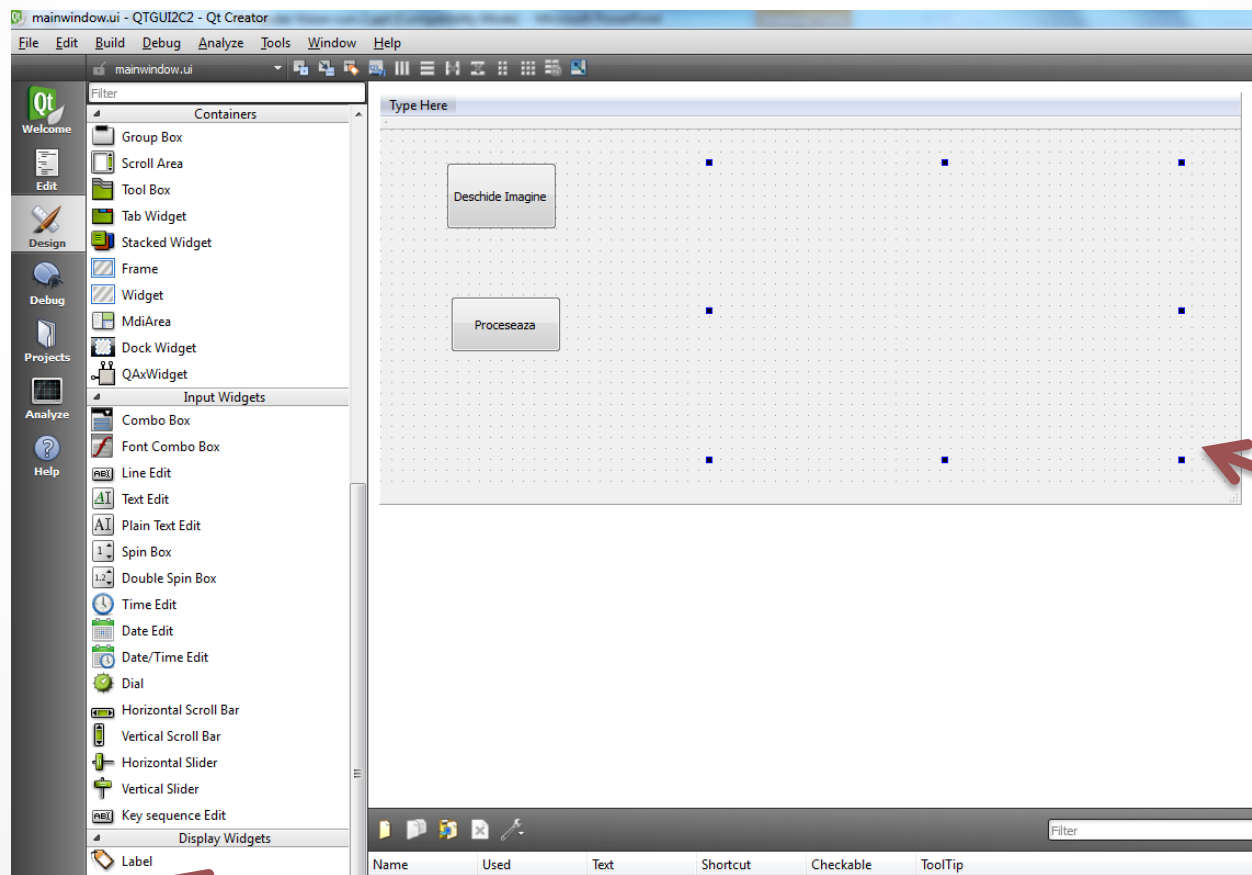
# Interfata in Qt pentru oglindirea unei imagini

- Primul buton deschide o fereastră de dialog care ne permite sa selectam fisierul.
- Dupa deschiderea pozei, al doilea buton deschide cea de a doua fereastră cu poza oglindita.



# O noua interfata in Qt pentru oglindirea unei imagini

- Cream un nou proiect in care adaugam si un label.



# O noua interfata in Qt pentru oglindirea unei imagini

- La acel label stergem textul prestabilit.
- Intentia este sa adaugam poza chiar in locul etichetei, in aceeaasi fereastră cu butoanele.
- Accesul catre eticheta se face prin intermediul atributului **label** al lui **ui**, adica **ui->label**.
- Pentru a adauga imaginea in fereastră Qt, trebuie sa facem o corespondenta intre QImage\* si Mat.
  - Este necesara inversarea celor 3 canale de culori din BGR in RGB.
  - Pentru aceasta se foloseste metoda cvtColor

\*Pentru mai multe detalii despre QImage, vedeti documentatia:

<http://qt-project.org/doc/qt-4.8/qimage.html>



# O noua interfata in Qt pentru oglindirea unei imagini

- In plus, dezactivam butonul **Proceseaza** pana cand imaginea nu este incarcata.
- Acest lucru il facem adaugand in constructorul MainWindow urmatoarea line de cod:
- `ui->pushButton_2->setEnabled(false);`
- In plus, la evenimentul pentru apasarea primului butonul, adaugam
- `if (poza.data)`  
`ui->pushButton_2->setEnabled(true);`

# O noua interfata in Qt pentru oglindirea unei imagini

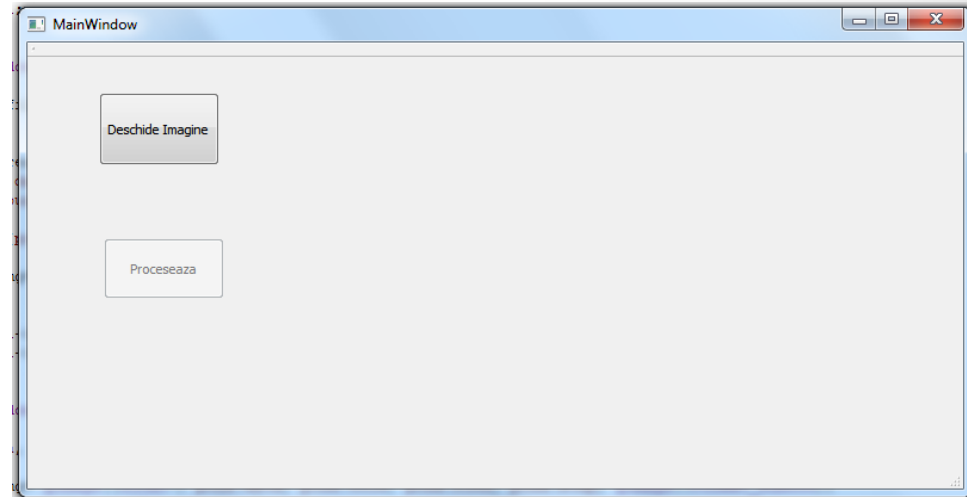
- Atributul **step** al obiectului poza contine lungimea in numarul de bytes a unei linii.
- In final, **mainwindow.cpp** va arata astfel:

```
mainwindow.cpp | MainWindow::on_pushButton_clicked(): void
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 MainWindow::MainWindow(QWidget *parent) :
5 QMainWindow(parent),
6 ui(new Ui::MainWindow)
7 {
8 ui->setupUi(this);
9 ui->pushButton_2->setEnabled(false);
10 }
11
12 MainWindow::~MainWindow()
13 {
14 delete ui;
15 }
16
17 void MainWindow::on_pushButton_clicked()
18 {
19 QString fileName = QFileDialog::getOpenFileName(this,
20 tr("Open image"), ".",
21 tr("Image Files (*.png *.jpg *.jpeg *.bmp)"));
22 poza= imread(fileName.toLatin1().data());
23 if (poza.data)
24 ui->pushButton_2->setEnabled(true);
25
26 cvtColor(poza, poza, CV_BGR2RGB);
27
28 QImage img= QImage((uchar*) poza.data, poza.cols,
29 poza.rows, poza.step, QImage::Format_RGB888);
30
31 ui->label->setPixmap(QPixmap::fromImage(img));
32 ui->label->resize(ui->label->pixmap()->size());
33 }
34
35 void MainWindow::on_pushButton_2_clicked()
36 {
37 flip(poza, poza, 1);
38
39 QImage img= QImage((uchar*) poza.data, poza.cols, poza.rows, poza.step, QImage::Format_RGB888);
40 ui->label->setPixmap(QPixmap::fromImage(img));
41 }
42 }
```



# O noua interfata in Qt pentru oglindirea unei imagini

- In prima faza, butonul **Proceseaza** este dezactivat, iar dupa incarcarea pozei acesta este activat.
- La apasarea sa, se obtine oglindirea pozei.



# Exercitii

1. Modificati proiectul anetrior astfel incat in afara de butonul de incarcare a pozei sa existe inca 3 alte butoane:
  - Unul pentru oglindire orizontala
  - Unul pentru oglindire verticala
  - Unul pentru ambele oglindiri
2. Pornind de la proiectul rezolvat, realizati o aplicatie care sa contina butonul de deschidere a unei imagini si unul care sa inverseze canalele de culori R, G si B.

# Cum functioneaza GUI-ul Qt

- Obiectele comunica prin semnale si slot-uri.
- Cand are loc un eveniment, un semnal este emis.
- Un slot este o metoda speciala care este apelata cand un semnal la care este conectata este emis.
- Slot-urile sunt definite in fisierul header.
- Accesul catre toate componentele in clasa principala `MainWindow` se face prin intermediul variabilei `ui` care este declarata in definitia clasei **`MainWindow`**.

# Rularea unui fisier video

- Citim frame-urile din fisier si le redam poza cu poza.
- Dupa fiecare afisare, adaugam o mica intarziere (delay).
  - Daca aceasta este mai mica, filmul merge in ffw

```
using namespace cv;
using namespace std;

int main()
{
 VideoCapture cap("D:/film.avi");

 if (!cap.isOpened())
 {
 cout << "Fisierul video nu poate fi rulat" << endl;
 return -1;
 }
 double fps = cap.get(CV_CAP_PROP_FPS); //luam frame-urile pe secunda din clip
 cout << "Frame-uri pe secunda : " << fps << endl;

 namedWindow("Clipul meu",CV_WINDOW_AUTOSIZE);

 while(1)
 {
 Mat frame;

 if (!cap.read(frame)) //daca nu se pot citi frame-urile
 {
 cout << "Nu se poate citi fisierul video" << endl;
 break;
 }

 imshow("Clipul meu", frame); //afisam frame-ul in fereastra "Clipul meu"
 //o mica intarziere de 30 de milisecunde intre frame-uri este necesara. De asemenea, stopam clipul si cu ESC
 if(waitKey(30) == 27)//daca este apasat ESC
 {
 cout << "Am incheiat clipul cu ESC" << endl;
 waitKey(2000);//asteapta 2 secunde
 break;
 }
 }
 return 0;
}
```

# Rularea clipului

(Global Scope) main()

D:\Dropbox\Work\Predare\Computer Vision\Programe\CameraC2\Debug\CameraC2.exe

Frame-uri pe secunda : 30.0003

```
17 (.cap_read(frame)) //daca nu
{
 cout << "Nu se poate citi t
 break;
}

imshow("Clipul meu", frame); //
//o mica intarziere de 30 de m
if(waitKey(30) == 27)//daca est
{
 cout << "Am inceiat clipul
 waitKey(2000);//asteapta 2
 break;
}
}
return 0;
```

100 %

Watch 1

| Name | Value |
|------|-------|
|      |       |

Clipul meu