Cekeikon e OpenCV:

Q: O que é Cekeikon?
R: Cekeikon é uma biblioteca de funções e classes escritas por mim em C++ para funcionar junto com OpenCV. Tem código aberto e qualquer um pode usar a biblioteca toda ou parte dela onde quiser, para fins científicos, educacionais e/ou comerciais.

Q: Por que você usa Cekeikon?
R1: A primeira resposta é: Deixo dentro de Cekeikon as funções/classes que desenvolvi alguma vez e que podem ser úteis em outros programas e/ou aos alunos. Isto permite reaproveitar o código que já desenvolvi assim como passar esse código aos alunos.
R2: A segunda resposta é "histórica": Há muitos anos atrás, eu usava uma biblioteca em C++ desenvolvida por mim chamada Proeikon. Quando OpenCV disponibilizou a interface C++, comecei a migrar os programas do Proeikon para OpenCV. Para que a tradução de um programa Proeikon para OpenCV fosse a mais simples possível, escrevi Cekeikon. Assim, um programa em Cekeikon fica semelhante ao equivalente em Proeikon.

Q: Qual é a vantagem aos alunos programarem em Cekeikon em vez de OpenCV puro?
R1: A primeira responsta é: Programando em Cekeikon, os alunos ganham acesso a funções/classes desenvolvidas por mim (além de todas as funcionalidades de OpenCV). Além disso, os programas das apostilas (que usam algumas funções feitas por mim) rodam sem fazer nenhuma adaptação. Basta dar "copy and paste". Não vejo nenhuma desvantagem em usar Cekeikon em vez de OpenCV puro. Cekeikon faz tudo o que OpenCV faz, e mais algumas coisas.
R2: A segunda resposta é: Cekeikon facilita a instalação de OpenCV, principalmente em Windows. Cekeikon permite utilizar OpenCV2 ou OpenCV3, bastando escrever na hora da compilação -v2 ou -v3. A compilação de programas fica mais simples. Cekeikon vem várias com bibliotecas (por exemplo TinyDNN para deep learning) integradas.

Q: Por que um programa escrita em Cekeikon parece muito diferente da versão em OpenCV puro?
R: Existem duas formas de escrever um programa C++ usando OpenCV: com ou sem template. A maioria dos programas de OpenCV disponíveis na internet não usa template. Cekeikon sempre usa template. Isto faz com que um programa Cekeikon pareça diferente da maioria dos programas de OpenCV que se encontra na internet. Porém, a diferença não é entre usar Cekeikon e usar OpenCV puro. A diferença é entre programar OpenCV usando ou não template.

Q: Se a maioria das pessoas programa OpenCV sem usar template, por que você usa template?
R: Vejo uma série de vantagens em usar template. Tentarei responder esta pergunta longamente no final desta apostila.

Q: Mesmo depois desta explicação, não gostaria de usar Cekeikon.
R: Os alunos não são obrigados a programarem em Cekeikon. Não precisam nem usar OpenCV. Podem usar outras bibliotecas de processamento de imagem. Só peço que programem em C++.

Q: Por que os alunos são obrigados a desenvolverem os programas de processamento de imagens em C++? Por que não uma outra linguagem como Python ou Matlab?
R: Os testes mostram que C/C++ é algo como 10 a 100 vezes mais rápido do que Python/Matlab:

- `http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp`
- `http://jonathankinlay.com/2015/02/comparison-programming-languages/`
- `https://julialang.org/benchmarks/`

As linguagens C e C++ possuem velocidades bem semelhantes, sendo C um pouquinho mais rápido que C++:

- `https://benchmarksgame.alioth.debian.org/u64q/c.html`

Em processamento de imagem, velocidade é essencial. Se você vai só chamar as funções que foram escritas por alguma outra pessoa (provavelmente em C/C++), não há muito problema em usar interface Python/Matlab. Agora, se você quer desenvolver as funções otimizadas de baixo nível, deve escrever em C/C++ (e talvez alguns trechos críticos em Assembler, Cuda, etc).

Q: Como faço para descobrir como usar as funções/classes do Cekeikon, se não há manual?
R: Esta apostila foi escrita para ensinar as funcionalidades básicas do Cekeikon. Outras apostilas dos cursos trazem vários exemplos de Cekeikon. Se quiser documentação mais detalhada ou quiser descobrir outras funcionalidades, você deve olhar o cabeçalho e o código do Cekeikon (no diretório cekeikon5/cekeikon/src). Para achar alguma função no código do Cekeikon, você pode usar o programa "grep". Por exemplo, se quiser descobrir onde está definida "dcReject" vá para diretório cekeikon5/cekeikon/src e escreva:

```
~/cekeikon5/cekeikon/src$ grep dcReject *.h
```

Uma vez descoberto que dcReject encontra-se em cekmm.h, abra esse arquivo e o arquivo .cpp correspondente, usando o seu editor preferido:

```
~/cekeikon5/cekeikon/src$ geany cekmm.h cekmm.cpp &
```

E localize dcReject. Em Linux, grep vem pré-instalada. Em Windows, grep é automaticamente instalada junto com Cekeikon.

Herança da biblioteca antiga Proeikon

Para que um programa em Cekeikon ficasse parecido com Proeikon, dei nomes alternativos a alguns tipos pré-existentes do C++/OpenCV. Os principais são:

```
typedef unsigned char GRY;
ou typedef uchar GRY;
ou typedef uint8_t GRY;
```

GRY é usado para armazenar um pixel em níveis de cinza. Vai de 0 a 255 e ocupa um byte. Equivale a unsigned char ou uchar ou uint8_t em C/C++.

```
typedef Vec3b COR;
```

COR é usado para armazenar um pixel em níveis de cinza. Consiste de 3 bytes que representam blue, green e red (o padrão do OpenCV é BGR e não RGB como nas outras bibliotecas). Equivale a Vec3b do OpenCV.

```
typedef float FLT;
```

FLT é usado para armazenar um pixel em níveis de cinza, em ponto flutuante. Consiste variável ponto flutuante de 4 bytes. Equivale a float do C/C++. Dentro do Cekeikon, implicitamente FLT vai de 0.0 (preto) até 1.0 (branco).

```
typedef complex<FLT> CPX;
```

CPX é usado para armazenar um pixel complexo. Consiste duas variáveis ponto flutuante de 4 bytes. Equivale a complex<float> do C++.

Outros tipos menos usados:

```
typedef short SHT;
typedef double DBL;
typedef Vec3f CORF;
typedef complex<DBL> CPXD;
```

Tipos definidos para ter em Linux definições equivalente de windows.h:

```
#ifdef __unix__
  // Para ter definicoes equivalentes a VC-Windows em Linux
  typedef uint8_t BYTE;
  typedef uint16_t WORD;
  typedef uint32_t DWORD;
  typedef uint64_t QWORD;
#endif
```

Com isso, temos os seguintes tipos principais de imagens:

Cekeikon	OpenCV puro	Descrição
`Mat_<GRY>`	`Mat_<unsigned char>`	imagem em níveis de cinza
`Mat_<COR>`	`Mat_<Vec3b>`	imagem colorida
`Mat_<FLT>`	`Mat_<float>`	imagem em níveis de cinza onde cada pixel é ponto flutuante de 0 a 1
`Mat_<CPX>`	`Mat_< complex<float> >`	imagem onde cada pixel é um número complexo

Programação por template do C++

Provavelmente, poucos alunos conhecem o que é um "template" de C++. Para poder explicar a diferença entre programar com ou sem template em OpenCV, vamos estudar rapidamente "template" de C++.

Digamos que o aluno queira escrever uma função que soma um numa variável. Só que a variável pode ser do tipo int ou double. Neste caso, o aluno teria que escrever duas funções:

```
void somaUm(int& v) {
  v++;
}

void somaUm(double& v) {
  v++;
}
```

Devido a um mecanismo chamado de “overloaded function name” de C++, se o aluno chamar:

```
int i=2; somaUm(i);
```

será chamada automaticamente a primeira função. Se chamar com uma variável double, será chamada automaticamente a segunda função.

O aluno foi escrevendo o seu programa, e notou que às vezes precisava somar um numa variável short, outra vez numa variável unsigned int, outra vez numa variável float, long long int, etc. Para cada tipo de variável, precisava escrever uma função diferente. Será que não tem um jeito de escrever uma única função somaUm que sirva para todos os tipos de variáveis? "Template" resolve este problema:

```
template<class T>
void somaUm(T& v) {
  v++;
}
```

Esta única função template serve para todos os tipos. Com esta função, pode-se escrever todas as linhas abaixo:

```
int i=2; somaUm(i);
float f=4.0; somaUm(f);
double d=5.0; somaUm(d);
unsigned int u=8; somaUm(u);
```

Agora, consideremos um outro problema. Digamos que o aluno quer ter uma classe “número complexo” (esta classe já existe em C++, mas vamos supor que não exista). Só que quer ter classe “complexo” onde as variáveis real e imaginária podem ser do tipo int ou double. Normalmente, teria que escrever 2 classes diferentes, correto?

```
class ComplexoInt {
  int real, imag;
  ...
};

class ComplexoDouble {
  double real, imag;
  ...
};
```

Usando “template”, é possível escrever uma única classe que serve para os dois casos:

```
template<class T>
class Complexo {
  T real, imag;
  ...
};
```

Com esta implementação, é possível trabalhar com complexo de int, float, double, short, long int, long long int, etc. Como é que diz se queremos trabalhar com int ou double? Fazendo:

```
Complexo<int> ci;
Complexo<double> cd;
Complexo<short> cs;
Complexo<float> cf;
```

Vocês (os alunos) já usaram este conceito (tipo com template) em classes do tipo:

```
vector<double> v;
queue<int> qu;
stack<char> s;
```

De forma semelhante, em Cekeikon/OpenCV, pode-se trabalhar com templates:
Mat_<GRY>
Mat_<COR>
Mat_<FLT>
Mat_<CPX>

Programação Cekeikon e OpenCV com/sem template

Agora, vamos voltar ao problema original. Existem duas formas diferentes de escrever programa C++ para OpenCV: usando template ou não. Cekeikon usa programação por template. Vamos pegar um programa Cekeikon muito simples, que gera a imagem negativa. Note o uso do "template" que acabamos de aprender.

```
//neg_cek.cpp
#include <cekeikon.h>
int main() {
   Mat_<GRY> a;
   le(a,"mickey_reduz.bmp");
   for (int l=0; l<a.rows; l++)
      for (int c=0; c<a.cols; c++)
         a(l,c)=255-a(l,c);
   imp(a,"neg_cek.pgm");
}
```

Como já disse, chamo "unsigned char" de GRY. Traduzindo para OpenCV com template:

```
//neg_ocv1.cpp
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main() {
   Mat_<unsigned char> a;
   a=imread("mickey_reduz.bmp",IMREAD_GRAYSCALE);
   for (int l=0; l<a.rows; l++)
      for (int c=0; c<a.cols; c++)
         a(l,c)=255-a(l,c);
   imwrite("neg_ocv1.pgm",a);
}
```

Repare que os dois programas são praticamente iguais. As três primeiras linhas do programa OpenCV estão incluídas no cabeçalho do Cekeikon. Porém, apareceu IMREAD_GRAYSCALE, que não havia no programa Cekeikon. A função imread precisa dele, pois não sabe o tipo de imagem onde vai ser armazenada a imagem lida. A função le não necessita desta informação, pois o tipo de imagem que se deseja ler está em template. A maioria dos exemplos de OpenCV usa a matriz sem template, ficando:

```
//neg_ocv2.cpp
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main() {
   Mat a;
   a=imread("mickey_reduz.bmp",IMREAD_GRAYSCALE);
   for (int l=0; l<a.rows; l++)
      for (int c=0; c<a.cols; c++)
         a.at<uchar>(l,c)=255-a.at<uchar>(l,c);
   imwrite("neg_ocv2.pgm",a);
}
```

A matriz ficou sem tipo. Isto é, o compilador não sabe se a imagem é de níveis de cinza, colorida, float ou complexo. Isso vai ser decidido durante a execução. Repare que apareceu `at<uchar>`. Como o compilador não sabe o tipo de matriz durante a compilação, o programador deve informar ao compilador qual é o tipo de matriz em todos os acessos à matriz. Acho isto muito ruim.

Por que a função "le" do Cekeikon não precisa de IMREAD_GRAYSCALE? Por que essa função chama internamente a versão correta do "imread" dependendo do tipo da imagem. Se mandar ler uma imagem colorida numa matriz colorida, faz o óbvio sem precisar dizer nada. O mesmo se mandar ler uma imagem em níveis de cinza numa matriz em níveis de cinza. Se mandar ler uma imagem colorida numa matriz em níveis de cinza, faz a conversão adequada automaticamente. O mesmo vale para outras combinações de tipos de imagens/matrizes (float, double, complexo, etc).

Em programação, quanto mais simples for o código, menos chance de errar. Cekeikon deixa o código mais limpo.

Um outro exemplo. Um programa que mostra na tela uma imagem vermelha com um traço preto.

```
//mostra_cek.cpp
#include <cekeikon.h>
int main() {
  Mat_<COR> a(100,100,COR(0,0,255));
  for (int i=0; i<a.rows; i++)
    a(i,i)=COR(0,0,0);
  mostra(a);
}
```

Vamos traduzir este programa para OpenCV puro sem template.

```
//mostra_ocv2.cpp
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;
int main() {
  Mat a(100,100,CV_8UC3);
  a.setTo(Vec3b(0,0,255));
  for (int i=0; i<a.rows; i++)
    a.at<Vec3b>(i,i)=Vec3b(0,0,0);
  imshow("janela",a);
  waitKey();
}
```

Chamo `Vec3b` de `COR`. Para alocar uma imagem 100x100 em OpenCV sem template, é necessário escrever "`CV_8UC3`". Pois, como a matriz é sem tipo, o compilador não sabe quanto de espaço é necessário alocar na hora da compilação. Cekeikon não precisa disso por que usa template. Além disso, OpenCV sem template precisa escrever `at<Vec3b>` toda vez que acessar um elemento da matriz. Cekeikon não precisa.

Resumindo, existem duas formas de programar OpenCV em C++: com template e sem template. A programação em Cekeikon (como ensino nas aulas) é quase igual a programar OpenCV com template. Programar OpenCV sem template ("`Mat a`") traz a necessidade de especificar o tipo de matriz (ex: imagem colorida) em várias partes do programa: `CV_8UC` na alocação, `IMREAD_COLOR` na leitura e `at<Vec3b>` no acesso. Programar em Cekeikon elimina essas necessidades: basta escrever uma vez "`Mat_<COR> a`".

Funções básicas de Cekeikon

1) Conversão de imagem. A função "converte" faz conversão "razoável" ou "natural" entre diferentes tipos de imagens. Na imagem GRY 0 é preto e 255 é branco. Na imagem FLT, 0 é preto e 1 é branco. Na imagem COR, COR(0,0,0) é preto e COR(255,255,255) é branco, na ordem BGR. Na imagem CPX, CPX(0,0) é preto e CPX(1,0) é branco.

```
EXPORTA void converte(Mat_<GRY> ent, Mat_<COR>& sai);
EXPORTA void converte(Mat_<COR> ent, Mat_<GRY>& sai);
EXPORTA void converte(Mat_<GRY> ent, Mat_<FLT>& sai, double alpha=1.0/255.0, double beta=0);
EXPORTA void converte(Mat_<FLT> ent, Mat_<GRY>& sai, double alpha=255, double beta=0);
EXPORTA void converte(Mat_<FLT> ent, Mat_<COR>& sai);
EXPORTA void converte(Mat_<COR> ent, Mat_<FLT>& sai);
EXPORTA void converte(Mat_<GRY> ent, Mat_<CPX>& sai);
EXPORTA void converte(Mat_<CPX> ent, Mat_<COR>& sai, char modo='h');
// modo=='h': converte MATCPX ent para MATCOR d usando HSV
// modo=='r': converte MATCPX ent para MATCOR d usando RB
EXPORTA void converte(Mat_<FLT> ent, Mat_<CPX>& sai);
```

2) Leitura de imagem

Cekeikon	OpenCV puro com template

`Mat_<COR> a;`
`le(a,"exemplo.ppm");`
 	
`Mat_<GRY> b;`
`le(b,"exemplo.pgm");`
	`Mat_<Vec3b> a;`
	`a=imread("exemplo.ppm",IMREAD_COLOR);`
	
	`Mat_<uchar> b;`
	`a=imread("exemplo.pgm",IMREAD_GRAYSCALE);`

No Cekeikon, não precisa especificar IMREAD_COLOR ou IMREAD_GRAYSCALE. A função lê faz a conversão "razoável", chamando a função "converte". Permite ler imagens tipo .TGA e .TXT. Permite ler Mat_<FLT> e Mat_<CPX> usando formato .IMG.

3) Escrita de imagem

Cekeikon	OpenCV puro com template

`Mat_<COR> a;`
`...`
`imp(a,"exemplo.ppm");`
 	
`Mat_<GRY> b;`
`...`
`imp(b,"exemplo.pgm");`
	`Mat_<Vec3b> a;`
	`...`
	`imwrite("exemplo.ppm",a);`
	
	`Mat_<uchar> b;`
	`...`
	`imwrite("exemplo.pgm",b);`

No Cekeikon, a especificação de qualidade de JPG e PNG pode ser feita acrescentando um número depois do sufixo. Permite gravar imagens tipo .TGA e .TXT. Permite gravar Mat_<FLT> e Mat_<CPX> usando formato .IMG.

4) Mostrar imagem na tela

Cekeikon	OpenCV puro com template

`Mat_<COR> a;`
`mostra(a);`
	`Mat_<Vec3b> a;`
	`namedWindow("janela");`
	`imshow("janela",a);`
	`waitKey(0);`

No Cekeikon, pode-se mostrar imagens tipo Mat_<FLT>, Mat_<CPX>.

Iremos vendo outras funções à medida em que aparecer necessidade.

Funções auxiliares de Cekeikon

1) Leitura dos argumentos:

```
EXPORTA void convArg(int& i, string arg);
EXPORTA void convArg(double& i, string arg);
EXPORTA void convArg(float& i, string arg);
EXPORTA void convArg(bool& b, string arg);
```

Exemplo:

```
  double brilho;
  if (sscanf(argv[3],"%lf",&brilho)!=1) xerro1("Erro: Leitura brilho");
```

Torna-se:

```
  double brilho;
  convArg(brilho,argv[3]);
```

Para variável booleano, palavra que começa com V, T ou 1 indica verdadeiro. Palavra que começa com F ou 0 indica falso.

2) Impressão colorida na tela:

```
//     0 = Preto        8 = Cinza
//     1 = Azul         9 = Azul claro
//     2 = Verde        A = Verde claro
//     3 = Verde-água   B = Verde-água claro
//     4 = Vermelho     C = Vermelho claro
//     5 = Roxo         D = Lilás
//     6 = Amarelo      E = Amarelo claro
//     7 = Branco       F = Branco brilhante
// f de 0 a 15
// b de 0 a 7 no Linux e 0 a 15 no Windows
EXPORTA void colorPrint(string st, int f=-1, int b=-1);
```

Imprime string st na tela com cor de foreground f e background b. Se não especificar f ou b, imprime com cores default.

3) Funções de debug (ajudam debugar o programa):

Antigos (use preferencialmente as novas funções):

```
  EXPORTA void debug(string s1, string s2="");
  EXPORTA void debug(int i);
```

Novos:

```
  #define xdebug ...
  #define xdebug1(st1) ...
  #define xprint(x) ...
```

Exemplos:

```
  xdebug; // Imprime nome do arquivo e o número da linha quando executa.
  xdebug1("Passei por aqui"); // Imprime nome do arquivo, número da linha e mensagem.
  int j=2; xprint(j); // imprime j=2
```

4) Funções para abortar programa:

Antigos (use preferencialmente as novas funções):

```
  EXPORTA void erro(string s1, string s2="", string st3="");
```

Novos:

```
  #define xerro ...  // Imprime nome de arquivo, número de linha e aborta.
  #define xerro1(st1) ... // Imprime nome de arquivo, número de linha, st1 e aborta.
```

Exemplos:

```
  xerro;
  xerro1("Erro: Parei aqui");
```

5) Funções para cronometrar tempo:

Antigos (use preferencialmente as novas funções):

```
  EXPORTA clock_t centseg();
  EXPORTA clock_t miliseg();
```

Novos:

```
  EXPORTA TimePoint timePoint(); // Registra tempo atual
   EXPORTA double timeSpan(TimePoint t1, TimePoint t2); // Calcula quantos segundos passaram
entre t1 e t2 e armazena em segundos
  #define impTempo(t1) ... // Imprime quantos segundos passaram de TimePoint t1 até agora
```

Exemplos:

```
  TimePoint t1=timePoint();
  //Faz algum processamento
  TimePoint t2=timePoint();
  double t=timeSpan(t1,t2); // t armazena segundos que passaram entre t1 e t2.
  impTempo(t1); // Imprime quantos segundos passaram de t1 até agora.
```