C++:

"static"关键字

- 修饰类中的成员方法，与其他非static的区别

修饰类中的成员方法：被static修饰的成员方法是类方法，不依赖于任何实例对象，可以直接通过类名调用。与其他非static的成员方法不同，static方法不能访问非静态成员变量和非静态成员方法，只能访问静态成员变量和静态成员方法。

- 修饰文件中的一个全局变量

修饰文件中的一个全局变量：被static修饰的全局变量只能在当前文件中被访问，不能被其他文件访问。这种静态变量也称为文件作用域变量。

- 修饰函数内的一个变量

修饰函数内的一个变量：被static修饰的局部变量称为静态局部变量，它的生命周期与程序运行期间一样长，即使函数返回，静态局部变量的值也会被保留。静态局部变量只会被初始化一次，下一次调用函数时会保留上一次的值。

- 传值和传引用有什么区别?

传值是指将参数的值复制一份，然后传递给函数，在函数内部修改参数的值不会影响到原始参数的值。这种方式适用于参数值较小或者不需要修改原始参数的情况。

传引用是指将参数的地址传递给函数，函数可以直接访问原始参数的值，修改参数的值会影响到原始参数的值。这种方式适用于参数值较大或需要修改原始参数的情况。但需要注意参数的作用域和生命周期。

- 传引用和传指针有什么区别?

传指针可以使用空指针，而传引用不允许使用空引用。

传指针是将参数的地址传递给函数，函数内部通过指针访问参数的值，

而传引用是将参数的引用传递给函数，函数内部直接访问参数的值。

传指针需要注意指针是否为空，指针是否越界等问题，而传引用不需要考虑这些问题，因此更加安全。

- C++新特性:c11-c14标准出的新东西
  智能指针分哪几种类型?

C++智能指针主要分为以下三种类型:

1. `unique_ptr`:独占式智能指针，一个`unique_ ptr`对象拥有对一个指针的独占权，当`unique_ptr`对象被销毁时，它所指向的对象也会被销毁。`unique_ ptr`不能被复制，但可以通过`std: :move()`函数转移所有权。
2. `shared_ ptr`:共享式智能指针，一个`shared_ ptr`对象可以有多个指针共享同一个对象，当最后一个`shared_ptr`对象被销毁时，它所指向的对象也会被销毁。`shared_ptr`使用引用计数来管理内存，可以通过`std : :make _ shared()`函数创建。

3. `weak_ ptr`:弱引用智能指针，一个`weak_ptr`对象指向一个由`shared_ptr`对象管理的对象，但不会增加引用计数。`weak_ptr`可以通过lock()函数获得一个指向对象的`shared_ptr`,当对象已经被销毁时，lock ()函数返回一个空的`shared_ptr`。

- `shared_ptr`使用引用计数来管理内存，是指在每个`shared_ptr`对象内部都维护了一个引用计数器，用于记录有多少个`shared_ptr`对象指向同一个对象。当一个新的`shared_ptr`对象被创建时，它会将引用计数器加1；当一个`shared_ptr`对象被销毁时，它会将引用计数器减1。当引用计数器的值为0时，表示没有任何`shared_ptr`对象指向该对象，此时该对象会被销毁。
- `weak_ ptr`可以通过lock()函数获得一个指向对象的`shared_ptr`，指向的对象是`weak_ptr`所指向的`shared_ptr`所管理的对象。如果`weak_ptr`所指向的`shared_ptr`已经被销毁，那么lock()函数会返回一个空的`shared_ptr`。
- 在使用`weak_ptr`时，需要注意不能直接通过`weak_ptr`访问所指向的对象，因为`weak_ptr`并不拥有该对象的所有权，只是一个弱引用。如果需要访问对象的成员变量或成员函数，应该先通过lock()函数获取一个指向对象的`shared_ptr`，然后通过该`shared_ptr`来访问对象的成员。如果lock()函数返回的`shared_ptr`为空，说明对象已经被销毁，此时不能再访问该对象的成员。

```
#include <iostream>
#include <memory>
class MyClass {
public:
MyClass() {
std::cout << "MyClass constructor" <<std::endl;
~MyClass() {
std::cout << "MyClass destructor" << std::endl;
void print() {
std::cout << "Hello, world!" << std::endl ;
};
int main() {
//使用unique_ptr创建对象
std::unique_ ptr<MyClass> ptr1(new MyClass());
ptr1->print();
//使用shared_ptr创建对象
std::shared ptr<MyClass> ptr2 = std::make_shared<MyClass>();
ptr2->print();
//使用weak_ ptr指向shared_ pt r所管理的对象
std: :weak_ptr<MyClass> ptr3 = ptr2;
//使用lock函数获得一个指向对象的shared_ptr
std::shared ptr<MyClass> ptr4 = ptr3.lock();
if (ptr4) {
ptr4->print();
else
std::cout << "Object has been dest royed"<< std::endl;
}
return 0;
}
```

std::vector arr;
int x;

//新的遍历方法

for(x:arr) {
  std::cout<<x;
}
std::string
std::unordered_map<std::string,std::unordered_map<int, std::future> >tasks;

- opencv: mat类，文件读写，缩放

OpenCV是一个跨平台的计算机视觉库，它提供了丰富的图像处理和计算机视觉算法，包括图像处理、特征提取、目标检测、目标跟踪、人脸识别、机器学习等领域。

Mat类
Mat类是OpenCV中用于表示图像的基本数据结构，它包含了图像的像素值、大小、通道数等信息。Mat类可以表示单通道或多通道的图像，支持多种数据类型，包括8位无符号整型、16位有符号整型、32位有符号或无符号整型、32位浮点型、64位浮点型等。Mat类还提供了丰富的操作函数，可以方便地对图像进行处理和操作。

OpenCV提供了`imread()`和`imwrite()`函数用于读取和保存图像文件。`imread()`函数 可以读取多种格式的图像文件包括JPEG、PNG、 BMP等格式，可以指定读取的图像类型和通道数。`imwrite()` 函数可以将图像保存为指定格式的文件，支持多种格式，包括JPEG、 PNG、 BMP等格 式。

OpenCV提供了resize()函数用于对图像进行缩放操作,可以按照指定的大小或缩放比例对图像进行缩放。resize()函数还可以指定插值方法，包括最近邻插值、双线性插值、双三次插值等方法。

```
#include<iostream>
#include<opencv2/opencv.hpp>
using namespace std;
using namespace cv;
int main(){
    Mat img =imread("test.jpg");
    if(img.empty()){
        cout<<failed<<endl;
    }
    Mat img_resized;
    resize(img,img_resized,Size(640,480),0,0,INTER_LINEAR);
//src：输入图像，可以是Mat类型或其他支持的图像类型。
//dst：输出图像，与输入图像大小和数据类型相同。
//dsize：输出图像的大小，可以是Size类型或其他支持的大小类型。
//fx：水平方向上的缩放比例，如果为0，则根据dsize计算缩放比例。
//fy：垂直方向上的缩放比例，如果为0，则根据dsize计算缩放比例。
//interpolation：插值方法，可以是INTER_NEAREST、INTER_LINEAR、INTER_CUBIC、
INTER_AREA、INTER_LANCZOS4等方法。

CV算法：
    imwrite("test_resizeed.jpg",img_resized);
    return 0;
}
```

- CV算法:

CV算法是计算机视觉中的一种算法，CV算法基于数学和统计学理论，通过对图像进行分析和处理，提取出有用的特征信息，从而实现图像的自动识别和处理。
\1. 数据预处理
首先需要对输入的图像进行预处理，包括图像的灰度化、归一化、去噪等操作。这些操作可以提高算法的鲁棒性（鲁棒性是指系统或算法在面对各种异常情况时仍能保持良好的性能和稳定性。鲁棒性是指算法对于图像中的噪声、遮挡、光照变化、姿态变化、尺度变化等因素的适应能力。）和准确性。
\2. 特征提取
接下来需要对图像进行特征提取，提取出能够区分不同人脸的特征信息。常用的特征提取方法包括Haar特征、LBP特征、HOG特征等。

\3. 特征匹配
特征提取后，需要对不同人脸的特征进行匹配，找出与输入图像最相似的人脸。常用的特征匹配方法包括欧式距离、余弦相似度、相交比等。
\4. 分类器训练
为了提高算法的准确性，需要使用训练数据对分类器进行训练。训练数据包括正样本和负样本，正样本是指包含人脸的图像，负样本是指不包含人脸的图像。常用的分类器包括SVM、KNN、神经网络等。

PS:

\1.Haar特征是一种基于图像亮度变化的特征提取方法，可以用于人脸识别、目标检测等领域。Haar特征是由矩形区域的亮度差值组成的，可以通过计算图像中矩形区域的亮度值和来得到。Haar特征具有平移不变性和尺度不变性等特点，可以有效地提取图像中的边缘、角点、纹理等特征。
\2. LBP特征
LBP特征是一种基于局部纹理的特征提取方法，可以用于人脸识别、纹理分类等领域。LBP特征是由图像中每个像素点周围的像素值比较结果组成的，可以通过比较像素值与中心像素值的大小关系来得到。LBP特征具有旋转不变性和灰度不变性等特点，可以有效地提取图像中的纹理特征。
\3. HOG特征
HOG特征是一种基于梯度方向的特征提取方法，可以用于人脸识别、目标检测等领域。HOG特征是由图像中每个像素点周围的梯度方向组成的，可以通过计算图像中梯度方向的直方图来得到。HOG特征具有旋转不变性和尺度不变性等特点，可以有效地提取图像中的边缘、角点等特征。

\1. 欧式距离
欧式距离是指在n维空间中，两个点之间的距离。在计算机视觉中，欧式距离常用于比较两个特征向量之间的相似度。欧式距离越小，表示两个特征向量越相似。

$$d(x,y) = sqrt(sum((xi - yi)^2))$$

\2. 余弦相似度
余弦相似度是指两个向量之间的夹角余弦值，它可以用来比较两个向量之间的相似度。余弦相似度越大，表示两个向量越相似。

$$cos(x,y) = (x \cdot y)/(||x|| * ||y||)$$其中，x和y表示两个向量，$x \cdot y$表示它们的点积，$||x||$和$||y||$分别表示它们的模长。

- LINUX命令：find, grep, cd,

find命令用于在指定路径下查找符合条件的文件或目录。常用的选项包括：
\ -name：按照文件名查找，支持通配符。
\ -type：按照文件类型查找，包括普通文件（f）、目录（d）、符号链接（l）等。

\-size:按照文件大小查找，支持+和-符号表示大于或小于指定大小。
\-mtime: 按照文件修改时间查找，支持+和-符号表示在指定时间之前或之后修改的文件。

```
find  -name "*.txt" //寻找后缀为txt的文件
```

grep命令用于在文件中查找指定的文本内容的行。常用的选项包括：
-i：忽略大小写。
-r：递归查找子目录。
-n：显示行号。
-w：匹配整个单词。
-v：反转匹配，显示不包含指定文本的行。

```
grep "hello" -i file.txt //寻找文件中包含忽略大小写的文本的行
```

linux开发环境：
gcc :编译器

- -c：只编译不链接，生成目标文件。
- -g：生成调试信息。
- -O：控制优化级别，包括-O0、-O1、-O2、-O3等。
- -Wall：开启所有警告信息。

gdb :调试器

常用的调试命令包括：

- run：启动程序。
- break：设置断点。
- print：打印变量的值。
- step：单步执行。
- next：跳过函数调用。

make文件：

复刻了一下：

```
CC=gcc
CFLAGS= -wall -g
TARGET =program
SRCS =main.c foo.c bar.c
OBJS=$(SRCS:.c=.o)
//编译器、编译选项、目标程序名、源文件名和目标文件名。其中，OBJS变量通过将SRCS中的.c后缀替换成.o后得到
$(TARGET):%(OBJS)
    $(CC) -o $@$^  //将所有的目的文件连接成一个可执行文件

% .o %.c
  $(CC)&(CFLAGS) -c$< -o //将所有源文件编译成目标文件。
```

cmake：

CMake是一个跨平台的自动化构建系统，可以用于生成各种编译器和IDE所需的构建文件（如Makefile、Visual Studio项目等）。CMake使用简单的语法来描述构建过程，可以自动生成构建脚本，从而简化了软件的构建和安装过程。

【C++】Cmake使用教程（看这一篇就够了）*cmake教程*隐居的遮天恶鬼的博客-CSDN博客

vim：常用的指令：
\1. 移动光标
- h：左移光标。
- j：下移光标。
- k：上移光标。
- l：右移光标。
- w：移动到下一个单词的开头。
- b：移动到上一个单词的开头。
- 0：移动到行首。
- $：移动到行尾。
- gg：移动到文件开头。
- G：移动到文件结尾。
\2. 插入和删除
- i：在光标处插入文本。
- a：在光标后插入文本。
- o：在光标下插入新行。
- x：删除光标处的字符。
- dd：删除整行。

-dG: 清空内容

- u：撤销上一步操作。
- Ctrl + r：重做上一步操作。
\3. 搜索和替换
- /text：向前搜索指定文本。
- ?text：向后搜索指定文本。
- n：查找下一个匹配项。
- N：查找上一个匹配项。
- :%s/old/new/g：全局替换old为new。
\4. 保存和退出
- :w：保存文件。
- :q：退出Vim。
- :wq：保存并退出Vim。
- :q!：强制退出Vim，不保存修改。

Gpu介绍：cuda

CUDA 用于利用GPU进行高性能计算。CUDA支持C/C+编程语言，提供了一系列的API和工具，可以使开发者更方便地利用GPU进行并行计算。

numpy与xtensor:
numpy简要教程：https://m.runoob.com/numpy/

xtensor主页:入门 — 张量文档 (xtensor.readthedocs.io)

流媒体ffmpeg：

FFmpeg是一款开源的音视频处理工具，可以用于录制、转码、剪辑、播放和流媒体传输等多种音视频处理任务。FFmpeg支持多种音视频格式和编解码器，可以在多个平台上运行，包括Windows、Linux、Mac OS 等。

ffmpeg常用命令

```
//视频转码
ffmpeg -i input.mp4 -c:v libx264 -c:a aac -b:v 1M -b:a 128K output.mp4
```
将输入文件input.mp4转换为输出文件output.mp4，使用libx264编码器进行视频编码，使用aac编码器进行音频编码，视频码率为1Mbps，音频码率为128kbps。

```
//视频剪辑
ffmpeg -i input.mp4 -ss 00:00:10 -t 00:00:30 -c:v copy -c:a output.mp4
```
将输入文件input.mp4从第10秒开始剪辑，剪辑时长为30秒，视频和音频不进行重新编码，直接复制到输出文件output.mp4中。

```
//视频截图
ffmpeg -i input.mp4 -ss 00:00:10 -frames:v 1 -s 640x480 output.jpg
```
将输入文件input.mp4在第10秒处截取一张640x480的图片，保存为输出文件output.jpg。

```
//音频转码
ffmpeg -i input.mp3 -c:a aac -b:a 128k output.m4a
```
将输入文件input.mp3转换为输出文件output.m4a，使用aac编码器进行音频编码，音频码率为128kbps。

```
//视频合并
ffmpeg -i input1.mp4 -i input2.mp4 -filter_com plex "xxxxx"output.mp4
```

多线程与线程池：
线程与进程有什么区别?

- 进程是操作系统中分配资源的基本单位，每个进程都有独立的地址空间、系统资源和文件描述符等；而线程是进程中的一部分，共享进程的地址空间和系统资源，但有自己的栈空间和线程上下文。
- 线程间的切换开销比进程间的切换开销要小得多。
- 线程共享进程的资源，多个线程可以并发执行，提高了程序的并发性和响应性；而进程间的通信需要使用IPC机制，开销较大。

Ps：IPC：进程间通信的一种机制，用于在不同的进程之间传递数据和信息。在现代操作系统中，进程间通信是非常常见的，例如多个进程共享同一份数据、交换消息等。

线程同步，互斥?

- 线程同步是为了协调多个线程之间的执行顺序和行为，以达到正确的结果；
- 而线程互斥是为了保证多个线程之间对共享资源的访问不会发生冲突。

  PS:常见实现：互斥锁用于线程互斥的机制，条件变量用于线程同步的机制。

版本管理工具 : git

Git教程 - 廖雪峰的官方网站 (liaoxuefeng.com)

常用git命令

c++类的构造函数

拷贝构造:深拷贝，浅拷贝

移动构造:用于创建一个新对象 并将其初始化为另一个对象的右值引用。移动构造函数通常用于提高程序的性能，避免不必要的复制操作。

赋值构造:一种特殊的成员函数，用于将一个对象的值赋给另一个对象。赋值构造函数通常用于实现对象的深拷贝，避免浅拷贝带来的问题。

重构 =符来实现赋值

```
class myclass{
public:
    myclass operator=(const myclass &temp){
        xxx=xx;
        xx=xx;
    }
};
```

问题：

1.我这边需要安装CUDA吧?

2.git应该需要深入学一下吧?

3.linux是不是要系统的学一下?

4.