# Lab5 report for CS205

SID:11911702
Name:郭一潼
Date:12.15 代码链接如下https://github.com/Godblessmycode1/CS205



# 介绍

## Project介绍

1. 使用c++
2. 支持不同数据类型
3. 矩阵类包含行列以及数据信息
4. 重载常用的运算符
5. 当使用copy assignment不使用深拷贝
6. 使用roi函数时，不使用深拷贝

## Project完成情况介绍

1. 通过使用模板类
2. 实现了内存安全
3. 实现了operator和roi时浅拷贝

## 开发环境

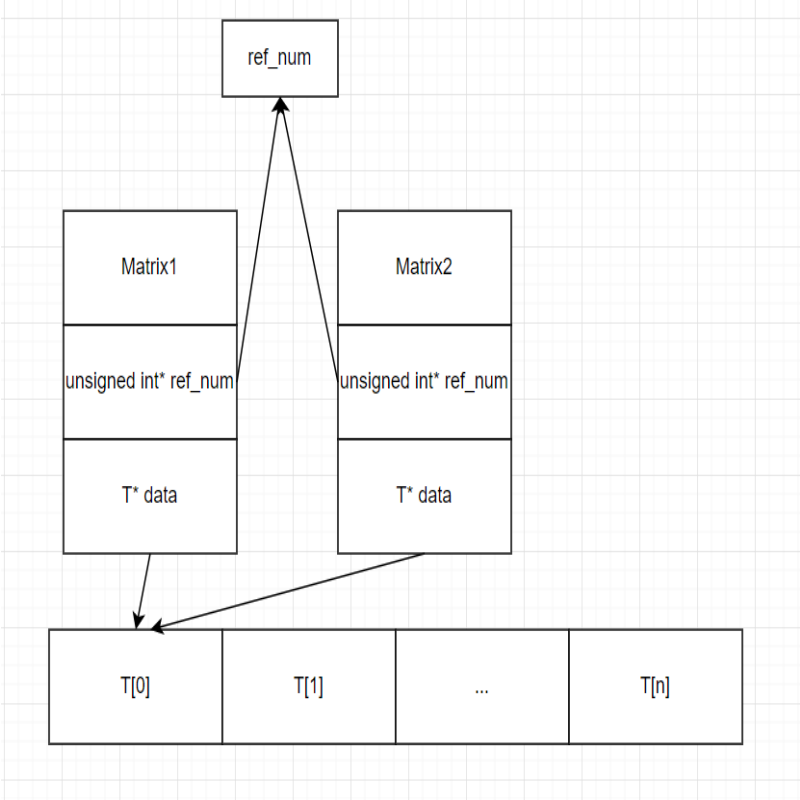
- x86_64
    - vscode (version 1.71)
    - WSL (version 2)
    - Ubuntu(22.04)
    - g++(11.2.0)

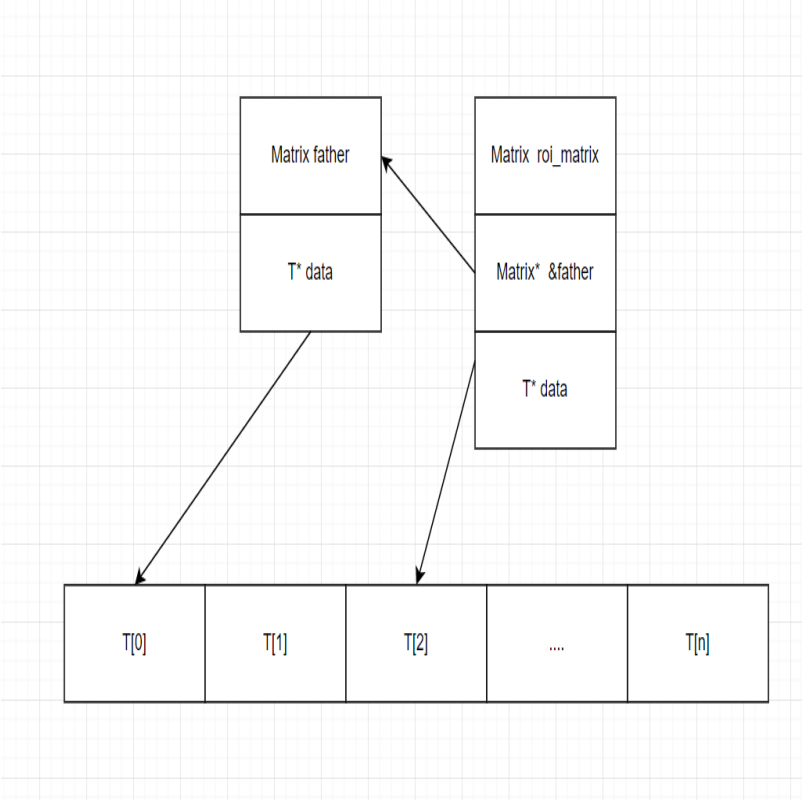
## 文件结构介绍

matrix.hpp是矩阵的source code， test.cpp是用于测验的cpp文件。

# 类设计思路

首先为了让矩阵支持多种数据类型，本次project选择使用模板类。
同时参考了opencv里的mat,**为了节省内存，防止实现memory hard copy,通过让多个通过 copy constructor和copy assignment 生成矩阵的数据指针指向同一内存**
考虑到roi区域也不能memory hard copy,我的设计思路是将**子矩阵的数据指针指向roi区域开始的数据位置。** 图示如下：






## 数据域介绍

```
template < class T>
class Matrix{
  private:
   size_t row;
   size_t col;
   size_t channel;
   unsigned int* ref_num;//用于解决多个矩阵指向一个地址 soft copy问题,只有在copy
assignment中才会调用。
   size_t step;//数据走一行多远。
   T* data;
   Matrix* parent;}
```

这里row,col是矩阵的行和列，channel是通道(>1)，**ref_num用于确定公用某一内存的矩阵的个数。因为涉及到roi区域（memory soft copy），因此会出现子矩阵的行的长度不等于col乘channel的状况，因此需要step来记录其数据一行有多长。Matrix类型指针则是用来矩阵是否为即其他矩阵的roi还是独立的矩阵，这里对于析构函数调用很有用。**

# 函数介绍

## 构造函数

- Matrix()
- Matrix(size_t row,size_t col,size_t channel,T* tp);
- Matrix(size_t row,size_t col,size_t channel);

- Matrix(const Matrix& mat);

其中前三个构造函数都会new unsigned int* ref_num 同时将它初始化为1, **只有copy constructor 会将数据copy过去，两矩阵的T*指向同一地址，同时将mat 的 ref_num[0] 加一。这样保证了每个独立矩阵的ref_num[0]和data一一对应，方便析构函数delete**。第三个构造函数则关注于创建连续的内存空间而不对其赋值。

```
Matrix<T>::Matrix(const Matrix<T> & mat){ //copy constructor
    this->row=mat.row; //复制行
    this->col=mat.col; //复制列
    this->channel=mat.channel; //复制通道。
    this->parent=mat.parent; //复制父亲
    this->step=mat.step; //复制一行长度
    this->ref_num=mat.ref_num;
    mat.ref_num[0]++;//引用数目加一
    this->data=mat.data;
}

template<class T>
Matrix<T>::Matrix(size_t row,size_t col,size_t channel,T* tp){
    if(channel==0){
      std::cerr<<"Channel should be at least 1,exit"<<std::endl;
      exit(EXIT_FAILURE);
    }
    this->channel=channel;
    this->row=row;
    this->col=col;
    this->step=col*channel;//一行数据多长。
    this->parent=nullptr;
    this->ref_num=new unsigned int{1};
    if(tp==nullptr){
      this->data=nullptr;
    }
    else{
      this->data=new T[row*col*channel];
      memcpy(this->data,tp,sizeof(T)*row*col*channel);
    }
}
```

## 析构函数

本次lab中会出现**多个矩阵指向同一个内存地址的情况，因此只有当内存地址没有任何一个矩阵指向的时候才释放数据。同时对于子矩阵，应当只delete ref_num,因为若delete []data,父矩阵调用析构函数的时候会出现内存被释放两次的问题，父矩阵既要delete ref_num,又要delete[] data**。

```
template<class T>
Matrix<T>::~Matrix(){
  this->ref_num[0]--;
  if(this->ref_num[0]==0){ //无矩阵指向内存，应该删除
```

```
    if(this->parent==nullptr){  //不为子矩阵才会删除数据，若为子矩阵则等待父亲的构析
函数即可。
      delete []this->data;
    }
    delete this->ref_num;//删除ref_num的内存//
  }
}
```

## 重载操作符

### 重载=运算符

重载 copy assignment, **本程序的copy assignment采用的方法是进行数据域的copy,因此需要注意当前矩阵的ref_num[0]是否为1，如果为1，则需要delete[]data以及delete this->ref_num，否则会造成内存泄露(内存数据未被回收，但没有矩阵可以访问)，否则this->ref_num[0]减一即可**

```
template <class T>
Matrix<T> Matrix<T>::operator =(const Matrix<T> & mat){
   if(this==&mat){
    return *this;
   }
   if(this->ref_num[0]==1){ //释放数据和指针。
    delete [] this->data;
    delete this->ref_num; //释放指针
   }
   else{

      this->ref_num[0]--;
   }
   this->row=mat.row;
   this->col=mat.col;
   this->step=mat.step;
   this->channel=mat.channel;
   this->data=mat.data;
   this->ref_num=mat.ref_num; //指向同一个ref_count并++
   this->ref_num[0]++;
   this->parent=mat.parent;
   return *this;
}
```

### 重载+运算符

本次project的+重载函数的返回值是一个新矩阵，它的每一个元素等于两个矩阵的对应元素和。

```
template<class T>
Matrix<T> Matrix<T>::operator+(const Matrix<T> & mat) const{
   if(this->row!=mat.row||this->col!=mat.col||this->channel!=mat.channel){
      std::cerr<<"The row or column or channel don't match please check carefully"
<<std::endl;
```

```
        exit(EXIT_FAILURE);
    }
    if (this->data==nullptr||mat.data==nullptr)
    {
     std::cerr<<"There is no data in at least one matrix"<<std::endl;
        exit(EXIT_FAILURE);
    }
    Matrix temp(this->row,this->col,this->channel);
  for(size_t i=0;i<this->row;i++){
      for(size_t j=0;j<this->col;j++){
        for(size_t k=0;k<this->channel;k++){
          temp.data[i*temp.step+j*temp.channel+k]=this->data[i*this->step+j*this-
>channel+k]+mat.data[i*mat.step+j*mat.channel+k];
        }
      }
     }
    return temp;
}
```

重载-运算符

本次project的-重载函数的返回值是一个新矩阵，它的每一个元素等于两个矩阵的对应元素差。

```
template <class T>
Matrix<T> Matrix<T>::operator-(const Matrix<T> & mat) const{
  if(this->row!=mat.row||this->col!=mat.col||this->channel!=mat.channel){
      std::cerr<<"The row or column or channel  don't match please check
carefully"<<std::endl;
      exit(EXIT_FAILURE);
    }
     if (this->data==nullptr||mat.data==nullptr)
    {
     std::cerr<<"There is no data in at least one matrix"<<std::endl;
      exit(EXIT_FAILURE);
    }
    Matrix temp(this->row,this->col,this->channel);
     for(size_t i=0;i<this->row;i++){
      for(size_t j=0;j<this->col;j++){
        for(size_t k=0;k<this->channel;k++){
          temp.data[i*temp.step+j*temp.channel+k]=this->data[i*this->step+j*this-
>channel+k]-mat.data[i*mat.step+j*mat.channel+k];
        }
      }
     }
    return temp;
}
```

重载*运算符

本次project的*重载函数的返回值是一个新矩阵，算法和矩阵乘法类似，唯一的区别在于多通道的矩阵乘法要分通道进行。

```
template <class T>
Matrix<T> Matrix<T>::operator*(const Matrix<T> & mat) const{
    if(this->row!=mat.col||this->col!=mat.row||this->channel!=mat.channel){
      std::cerr<<"The size of two matrixs don't match"<<std::endl;
      exit(EXIT_FAILURE);
    }
     if (this->data==nullptr||mat.data==nullptr)
   {
    std::cerr<<"There is no data in at least one matrix"<<std::endl;
      exit(EXIT_FAILURE);
   }
    Matrix temp(this->row,mat.col,this->channel);//带通道的矩阵相乘要一个通道一个通道的算
    //Eg.储存结果矩阵的第一行与第一列的channel1的结果是第一行与第一列channel1的结果.
    for(size_t row_index=0;row_index<this->row;row_index++){
        for(size_t col_index=0;col_index<mat.col;col_index++){
          for(size_t col_number=0;col_number<this->col;col_number++){
            for(size_t channel_index=0;channel_index<channel;channel_index++){

temp.data[row_index*temp.step+col_index*temp.channel+channel_index]+=this->data[row_index*this->step+col_number*this->channel+channel_index]*mat.data[col_index*mat.channel+col_number*mat.step+channel_index];
            }
          }
        }
    }
    return temp;
}
```

重载==运算符

如果两个矩阵都是有数据的话，则逐个通道检查数据是否相等，否则若两个都为空矩阵(mat.data==nullptr),则返回true.

```
template <class T>
bool Matrix<T>::operator==(const Matrix & mat){
    if(this->row!=mat.row||this->col!=mat.col||this->channel!=mat.channel){
      return false;
    }
    if(this->data!=nullptr&&mat.data!=nullptr){
       for(size_t i=0;i<this->row;i++){
        for(size_t j=0;j<this->col;j++){
          for(size_t k=0;k<this->channel;k++){
            if(this->data[(i*col+j)*channel+k]!=mat.data[(i*col+j)*channel+k]){
              return false;
            }
```

```
            }
          }
        }
    return true;
    }
    else{
      if(this->data==nullptr&&mat.data==nullptr){
        return true;
      }
      else{
        return false;
      }
    }
}
```

## roi区域函数

roi函数函数设计主要是4个参数，第一个参数是roi起始位置的row_index,第二个是roi区域的col_index,第三个是roi区域横向长度,第四个是roi区域纵向长度，**通过将子矩阵的data指向父矩阵的(row_index,col_index)位置，同时将子矩阵的row设置为纵向长度，将col设置为横向长度,将子矩阵的parent指针指向父矩阵。**

```
template <class T>
Matrix<T> Matrix<T>::roi(size_t row_index,size_t col_index,size_t
roi_row_length,size_t roi_col_length){
          if(this->data==nullptr){
            std::cerr<<"There is no data in the father matrix, return a matrix
with no data"<<std::endl;
            Matrix temp;
            return temp;
          }
          else if(row_index+roi_row_length>this-
>row||col_index+roi_col_length>this->col){
            std::cerr<<"The roi region is not valid"<<std::endl;
            exit(EXIT_FAILURE);
          }
          else{
              //开始提取roi,将子矩阵的data指针指向父矩阵的roi开始位置。
            Matrix temp;
            temp.channel=this->channel;
            temp.row=roi_row_length;
            temp.col=roi_col_length;
            temp.parent=this;
            temp.data=this->data+(row_index*this->step+col_index*channel); //父矩
阵row_index行,col_index列的位置指针。
            temp.step=this->step;//因为是soft copy，对于子矩阵来说，想要跳到下一行的
数据的长度与父矩阵的行长度相同。
            temp.ref_num=new unsigned int{1};
            return temp;
          }
}
```

# 内存安全原因

**对构造函数来说,所有ref_num与data都是一一对应的(非copy constructor的构造方法都是new ref_num以及new [] data都是一一对应的，copy constructor因为直接copy 参数矩阵的数据，同时在参数矩阵的ref_num加一)。同时在使用copy assignment时，进行了原矩阵判断ref_num[0]是否为一，若为一则先delete ref_num以及delete[]data，copy数据后的矩阵，ref_num与data依旧是一一对应的。因此通过构造函数以及copy assignment时不会出现内存泄露状况。**

# 总结

感觉这次lab的难点在于如何设计这个类，才能支持memory soft copy的情况下内存不泄露。 通过这次lab,感觉自己写的话，cpp内存管理还是蛮麻烦的，感觉c++用好真难，同时要小心谨慎。对操作符重载这些也更熟悉了。同时感觉模板类的方法具有普遍性，但不具有针对性，对于特定类型的矩阵运算以及优化，感觉不容易，算是工程上的trade off。