CPP_Project5_Matrix_Class@Jiko

- **Name：纪可鸣**
- **SID：12112813**

仓库网址： https://github.com/JikoSchnee/SUSTECH_CS205_C-CPP_Project/tree/main/Project5

写在前面

本次project因为在有限时间内没有找到很好地能兼顾 `兼容性强` (譬如两个不同数据类型的矩阵做运算)和 `代码简便` (尽量减少重复的工作)的方法，因此分别使用非模板类和模板类两种方法来实现。

※受篇幅限制，Project4中已出现过的错误处理、乘法加速等会一笔带过或者只在代码演示中呈现，并不会单独拎出来一个板块介绍。

※因为很多的思路是及其相似的，因此相同思路的仅挑选部分来介绍。

A.非类模板Matrix

写在前面

> **优点：**
>
> 兼容性更强(可以自动地在类型之间进行切换)；适合编程水平较低的用户，大部分情况下需要考虑的东西少，等等。
>
> **缺点：**
>
> 代码冗长而重复；更新其他数据或其他函数工作量大；不规范使用可能存在精度丢失，等等。

核心思想

因为矩阵所需的数据类型的个数是在可控范围内可数的，例如本题要求的 `unsigned char, short, int, float, double` 仅5个，因此考虑让五个类型的指针都存在，但同时只有一个指针根据矩阵的类型(`type`)被分配内存，后续根据 `type` 的变化会同时更新指针的分配：

```
1       size_t rows;
2       size_t cols;
3       size_t type;
4       unsigned char * ucP;    //type1
5       short * shortP;         //type2
6       int * intP;             //type3
7       float * floatP;         //type4
8       double * doubleP;       //type5
```

函数一览

```
//Constructor
explicit JChannel(size_t ROW = 1, size_t COL = 1, unsigned char *dataP =
nullptr);
explicit JChannel(size_t ROW = 1, size_t COL = 1, short *dataP = nullptr);
explicit JChannel(size_t ROW = 1, size_t COL = 1, int *dataP = nullptr);
explicit JChannel(size_t ROW = 1, size_t COL = 1, float *dataP = nullptr);
explicit JChannel(size_t ROW = 1, size_t COL = 1, double *dataP = nullptr);
//Destructor
~JChannel();
//Getter And Setter
size_t getRow();
size_t getCol();
char getType();
unsigned char getUCData(size_t row,size_t col);
short getShortData(size_t row,size_t col);
int getIntData(size_t row,size_t col);
float getFloatData(size_t row,size_t col);
double getDoubleData(size_t row,size_t col);
void setData(size_t row, size_t col, unsigned char key);
void setData(size_t row, size_t col, short key);
void setData(size_t row, size_t col, int key);
void setData(size_t row, size_t col, float key);
void setData(size_t row, size_t col, double key);
//Operator
JChannel operator+(JChannel const &Mat) const;
JChannel operator-(JChannel const &Mat) const;
JChannel operator*(JChannel const &Mat) const;
JChannel operator/(JChannel const &Mat) = delete;
JChannel & operator=(JChannel const &Mat);
bool operator==(JChannel const &Mat) const;
JChannel operator+=(JChannel const &Mat) = delete;
JChannel operator-=(JChannel const &Mat) = delete;
```

部分演示

Constructor

思路详解

五个类型分别对应五个构造器，根据传入数组的指针类型自动进行选择。例如传入的是一个double类型的指针，那么定义 `size_t type` 为对应的 `5` (对应数字参考 *C.零散的点---类型之间的转换*)，根据 `type` 来判定需要分配内存的指针是哪一个，分配内存并赋值。

代码演示

以double类型为例

```
JChannel::JChannel(size_t ROW, size_t COL, double *dataP) {
    rows = ROW;
    cols = COL;
    if (ROW == 0 || COL == 0) {
        std::cerr << "Error: Invalid size." << std::endl;
        if (abortIfError)std::abort();
    }
```

```
        if (!dataP) {
            std::cerr << "Error: Input a nullptr data pointer." << std::endl;
            if (abortIfError)std::abort();
        }
        type = 5;
        try {
            if (type == 1) {
                ucP = new unsigned char[ROW * COL * sizeof(unsigned char)];
            } else if (type == 2) {
                shortP = new short[ROW * COL * sizeof(short)];
            } else if (type == 3) {
                intP = new int[ROW * COL * sizeof(int)];
            } else if (type == 4) {
                floatP = new float[ROW * COL * sizeof(float)];
            } else if (type == 5) {
                doubleP = new double[ROW * COL * sizeof(double)];
            } else {
                std::cerr << "Error: Invalid data type." << std::endl;
                if (abortIfError)std::abort();
            }
        }
        catch (std::bad_alloc &ba) {
            std::cerr << ba.what() << std::endl;
        }
        try {
            if (type == 1) {
                memcpy(ucP, dataP, ROW * COL * sizeof(unsigned char));
            } else if (type == 2) {
                memcpy(shortP, dataP, ROW * COL * sizeof(short));
            } else if (type == 3) {
                memcpy(intP, dataP, ROW * COL * sizeof(int));
            } else if (type == 4) {
                memcpy(floatP, dataP, ROW * COL * sizeof(float));
            } else if (type == 5) {
                memcpy(doubleP, dataP, ROW * COL * sizeof(double));
            } else {

            }
        }
        catch (std::exception e) {
            std::cerr << e.what() << std::endl;
            delete[] dataP;
        }
    }
```

Getter and Setter

思路详解

对应的矩阵需要对应的getter，如果对一个double类型的矩阵使用了int类型的get，那么会返回错误提示(并终止程序)，反之才会返回正确的值。

对应矩阵的setter跟构造器思路类似，先检测输入的类型是否与矩阵匹配，而后再对其对应位置赋值。

代码演示

以double为例

```
double JChannel::getDoubleData(size_t row, size_t col) {
    if (type != 5) {
        std::cerr << "Error: The matrix type is not double." << std::endl;
        if (abortIfError)std::abort();
        return 0;
    }
    return doubleP[(row - 1) * cols + (col - 1)];
}
```

Operator*

思路详解

通过获取结果矩阵的类型(详见 *C.零散的点---类型之间的转换*)，根据不同的类型获得结果矩阵的数组，并依次创建矩阵。

代码演示

省略了重复的部分

```
JChannel JChannel::operator*(const JChannel &Mat) const {
    if (rows != Mat.cols || cols != Mat.rows) {
        std::cerr << "Error: Invalid to multiply this two matrix." << std::endl;
        if (abortIfError)std::abort();
    }
    size_t r = rows;
    size_t c = Mat.cols;
    size_t resultType = type > Mat.type ? type : Mat.type;
    if (type == 1) {
        unsigned char *data = nullptr;
        try {
            data = new unsigned char[r * c * sizeof(unsigned char)];
        } catch (std::bad_alloc &ba) {
            std::cerr << ba.what() << std::endl;
        }
        unsigned char tem = 0;
#pragma omp parallel for num_threads(16)
        for (int i = 0; i < r; ++i) {
            for (int k = 0; k < r; ++k) {
                tem = this->ucP[i * r + k];
                for (int j = 0; j < r; ++j) {
                    data[i * r + j] += tem * Mat.ucP[k * r + j];
                }
            }
        }
        JChannel result = JChannel(r, c, data);
        delete[] data;
        return result;
    }
    //...
    } else {
        std::cerr << "Error: Multiplying an unknown type matrix." << std::endl;
        if (abortIfError)std::abort();
```

```
    }
}
```

Operator+

思路详解

获取结果矩阵的类型，开一个tem来存放单个数据，一个结果数组来存放所有数据，最后根据两个矩阵的类型分别加上对应的数据，再根据这一数组来创建矩阵并返回。

代码演示

省略了重复部分

```
JChannel JChannel::operator+(const JChannel &Mat) const {
    if (rows != Mat.rows || cols != Mat.cols) {
        std::cerr << "Error: Adding two matrix whose rows and cols are not match." << std::endl;
        if (abortIfError)std::abort();
    }
    if (type != Mat.type && accuracyWarning) {
        std::cerr << "Warning: Adding two different types of matrix may cause accuracy loss." << std::endl;
    }
    size_t resultType = 0;
    resultType = type > Mat.type ? type : Mat.type;
    if (resultType == 0) {
        std::cerr << "Error: Adding result is an unknown type matrix." << std::endl;
        if (abortIfError)std::abort();
    }
    if (resultType == 1) {
        unsigned char *data = nullptr;
        try {
            data = new unsigned char[rows * cols * sizeof(unsigned char)];
        }
        catch (std::bad_alloc &ba) {
            std::cerr << ba.what() << std::endl;
            if (abortIfError)std::abort();
        }
        for (int i = 0; i < rows * cols; ++i) {
            data[i] = 0;
        }
        if (type == 1) {
            for (int i = 0; i < rows * cols; ++i) {
                data[i] += ucP[i];
            }
        } else {
            //...
        }
        if (Mat.type == 1) {
            for (int i = 0; i < rows * cols; ++i) {
                data[i] += Mat.ucP[i];
            }
        } else {
            //...
        }
```

```
            JChannel result = JChannel(rows, cols, data);
            delete[] data;
            return result;
        }
        } else {
            //...
        }
    }
```

B.类模板Matrix

> **优点:**
>
> 通过类模板可以直接适用更多数据类型，例如long、char等等；优化与更新方便，等等。
>
> **缺点:**
>
> 在速度优化以及遇到两个不同类型矩阵做运算的时候所需的工作量特别大，实现难度也大，等等。

写在前面

核心思想

在正规的操作中，不同类型的矩阵之间的运算是不应该存在的(存在精度丢失)，因此在保证使用者水平的情况下，并不需要兼容两个不同类型的矩阵之间运算的情况的，因此在模板类Matrix中并没有考虑这种情况，而是将经理集中在模板上。

操作符重载

operator=

使用了`std::shared_ptr`来避免`hardcopy`

```
JChannel2 &operator=(const JChannel2 &mat) {
        auto sp = std::shared_ptr<JChannel2<T>>(mat);
        return sp;
    }
```

operator==

```
bool operator==(const JChannel2 &mat) {
    if (row != mat.row||column!=mat.column)return false;
    for (int i = 0; i < row*column; ++i) {
        if (this->data[i]!=mat.data[i])return false;
    }
    return true;
}
```

operator<<

Problem Occured:

由于矩阵类是一个模板类，所以其函数也需要使用模板，但在重载 `<<` 时函数又应为友元函数，在这样神奇的组合下会出现错误提示：

`friend declaration 'std::ostream& operator<<(std::ostream&, const JMatrix<dataType>&)' declares a non-template function [-Wnon-template-friend]`

Problem Solving:

> **这类友元函数通常带有参数，并且参数中含有模板类定义的类型变量，例如：friend void Fn(MyClass &n)。由于我们不将该函数视为模板函数，因此对模板类的每个实例化版本都需要提供该函数的一个重载版本。**

```
1   std::ostream &operator<<(std::ostream &os, const JMatrix<int> &t) {
2       os<< typeid(t.data[0]).name()<<" int"<<" matrix with "<<t.row<<" rows and "
    <<t.column<<" columns and "<<t.channel<<" channels.";
3       return os;
4   }
```

~~以int类型为例，一个打印最基本信息的函数这样生成，其他n-1种类型如法炮制，但这样如果要修改文字内容的话就需要修改n次太过繁杂，因此这里使用了一个 `toString()` 函数来把基本信息转换为string类型。(由于不会被用户使用到，因此 `toString()` 并没有在头文件中声明)~~

但是我发现不把函数的声明和定义分开到 `.cpp` 和 `.h` 中，直接在类里面写就不会出现这个问题(?)，仔细一想貌似一些比较短的函数也没有分开写的必要，所以自此决定把大部分函数例如运算符重载以及一些简单的 `get` 函数都直接写在类里面了。

C.零散的点

通道

根据**网页**的解读，我的理解是以往理解的矩阵是一个二维平面，而通道数描述了平面的个数，构成一个三维的矩阵，因此将以往理解的矩阵称为JChannel，将矩阵用另一个类JMatrix嵌套JChannel实现：

```
1   class JMatrix {
2   private:
3       size_t rows;
4       size_t cols;
5       size_t chans;
6       size_t type;
7       JChannel * channels[4] = {nullptr, nullptr, nullptr, nullptr};
8   public:
9       JMatrix(size_t row = 1,size_t col = 1,size_t channel = 1,size_t type = 5){
10          rows = row;
11          cols = col;
12          chans = channel;
13          this->type = type;
14      }
15      void setChannel(size_t num, JChannel * channel){
16          if(num > chans||num < 1){
```

```
            std::cerr << "Error: Setting channel out of index." << std::endl;
            return;
        }
        if(rows!=channel->getRow()||cols!=channel->getCol()){
            std::cerr << "Error: Setting channel whose rows of columns are not match." << std::endl;
            return;
        }
        if(type!=channel->getType()){
            std::cerr << "Error: Setting channel whose type is not match." << std::endl;
            return;
        }
        channels[num-1] = channel;
    }
    unsigned char getUCData(size_t channel, size_t row,size_t col);
    short getShortData(size_t channel, size_t row,size_t col);
    int getIntData(size_t channel, size_t row,size_t col);
    float getFloatData(size_t channel, size_t row,size_t col);
    double getDoubleData(size_t channel, size_t row,size_t col);
    void setData(size_t channel, size_t row, size_t col, unsigned char value);
    void setData(size_t channel, size_t row, size_t col, short value);
    void setData(size_t channel, size_t row, size_t col, int value);
    void setData(size_t channel, size_t row, size_t col, float value);
    void setData(size_t channel, size_t row, size_t col, double value);
};
```

调用JChannel的函数例子：

```
double JMatrix::getDoubleData(size_t channel, size_t row, size_t col) {
    if (channel<1||channel>chans){
        std::cerr << "Error: Getting channel out of index." << std::endl;
        if (abortIfError)std::abort();
    }
    if (!channels[channel-1]){
        std::cerr << "Error: Getting channel that is empty." << std::endl;
        if (abortIfError)std::abort();
    }
    return channels[channel-1]->getDoubleData(row,col);
}
```

类型之间的转换

遵循c++自带的隐式转换原则，利用auto得到5个数据的10个不同组合转换后的结果：

```
auto type = type1 a + type2 b;
```

```
unsigned char + short = i
unsigned char + int = i
unsigned char + float = f
unsigned char + double = d
short + int = i
short + float = f
short + double = d
int + float = f
int + double = d
float + double = d
```

五个类型对应的type数字分别为1，2，3，4，5，因此刚好在两个矩阵运算时，取type较大的那一个作为结果的type即可。（如int+float=float，int为3，float为4，即在3和4中取最大值4，也就是float作为结果的类型）

错误流提示

```
        std::cerr << "Error: ..." << std::endl;
        if (abortIfError)std::abort();
```

```
std::cerr << "Warning: Adding two different types of matrix may cause accuracy loss."
          << std::endl;
```

使用了标准错误流输出来报错，并且仿照 `assert` 的形式(虽然不显示assert是在编译层面而这里是在运行层面)，可以通过把全局变量 `abortIfError` 改成 `false` 来让程序即使遇到错误也会继续运行下去；同理可以把 `accuracyWarning` 改成 `false` 来让程序不提示存在精度丢失的情况。

try catch

在申请内存的时候均使用了 `try catch` 函数

```
    try {
        if (type == 1) {
            ucP = new unsigned char[ROW * COL * sizeof(unsigned char)];
        } else if (type == 2) {
            shortP = new short[ROW * COL * sizeof(short)];
        } else if (type == 3) {
            intP = new int[ROW * COL * sizeof(int)];
        } else if (type == 4) {
            floatP = new float[ROW * COL * sizeof(float)];
        } else if (type == 5) {
            doubleP = new double[ROW * COL * sizeof(double)];
        } else {
            std::cerr << "Error: Invalid data type." << std::endl;
            if (abortIfError)std::abort();
        }
    }
    catch (std::bad_alloc &ba) {
        std::cerr << ba.what() << std::endl;
    }
```

D.总结

类模板虽然能隐式地实现更多种情况，但是在不同模板类之间的交互和乘法的加速就变得非常困难，而不使用类模板在除了交互之外的函数实现都需要许多重复，仅适用情况少的时候(例如本题要求的五个数据类型)。也许嵌套的类模板能够结合二者的优点，但经过多次尝试仍然没有办法实现在一个类中含两个或以上typename的情况，因此最终决定两个方法都写一遍体会一下差别。

最终使用类模板的代码仅几百行，不使用类模板的已经近千行，~~而且写完不使用类模板的完全不想再看再碰子~~，希望能在未来的学习中能解决这个问题。

转眼一学期过去已经是最后一次Project了，想当初听闻于老师CPP课的“恐怖”以及第一次project的洗礼几度欲放弃，幸好最后坚持下来了，在后来的日子里不仅深刻感受到于老师课程的有趣，同时也学到了很多非常有用的东西。总之非常感谢于老师这一学期的教导！