C++ 基础与深度解析
Project Final思路分享

助教 陈志伟

纲要

- 整体思路
- 实现方法

整体思路

- 本次作业主要考察C++泛型编程的相关知识点，主要涉及C++模板的相关知识。
- 本次作业主要是设计一个简单的矩阵运算库，涉及一些矩阵运算的相关操作。

基本要求

- 基本要求需要设计一个矩阵类，该类的成员变量如下：

```
std::vector<T> matrix_num;
int rows_ = 1;
int cols_ = 1;
```

- 同时需要支持三种构造的方式，即无参构造、初始化列表构造和指定矩阵尺寸大小构造：

```
Matrix(const unsigned int& rows = 1, const unsigned int& cols = 1);
Matrix(const std::initializer_list<T>& v);
```

基本要求

- 通过[][]进行访问：使用操作符[]重载

```
template <class T>
typename std::vector<T>::iterator Matrix<T>::operator[](std::size_t idx)
{
    return matrix_num.begin() + (idx * cols_);
}
```

- 矩阵运算：

```
Matrix operator+(Matrix& mat);
Matrix operator-(Matrix& mat);
Matrix operator*(Matrix& mat);
```

基本要求

- reshape,at,push_back：正常对设计的vector进行操作

```
/**
 * @brief  按照位置访问元素
 * @param n 元素位置
 * @return  元素值
 */
T& at(const unsigned int& n);
T at(const unsigned int& n) const;

/**
 * @brief  向矩阵插入元素
 * @param  member 插入元素
 * @return  void
 */
void push_back(const T member);

/**
 * @brief  修改矩阵尺寸
 * @param rows 行
 * @param cols 列
 * @return  void
 */
void reshape(const unsigned int& rows, const unsigned int& cols);
```

基本要求

- 异常抛出

```
template <class T>
T Matrix<T>::at(const unsigned int& n) const
{
    if (n >= matrix_num.size())
        throw "Matrix subscript out of bounds";
    return matrix_num[n];
}
```

- 异常捕获

```
try
{
    TestNormalMatrix();
    TestExtendMAtrix();
}
catch (char const* str)
{
    std::cout << "ERROR: " << str << std::endl;
}
```

扩展1

- 模板参数检查

```
template <class T>
concept isAddable = requires(T a, T b)
{
    a + b;
};
```

```
static_assert(isAddable<T>, "This type is not support addition ");
```

扩展2

- 需要实现静态内存分配，其成员变量

```
std::array<T, _Rows * _Cols> matrix_num;
static constexpr int rows_ = _Rows;
static constexpr int cols_ = _Cols;
```

- 对于乘法的操作符重载需要使用类模板的成员函数模板

```
template <int _Rows2, int _Cols2>
MatrixEx<T, _Rows, _Cols2> operator*(MatrixEx<T, _Rows2, _Cols2>& mat);
```

扩展3

- 矩阵拼接技巧

```
template <int _Type, class T, int _Rows1, int _Cols1, int _Rows2, int _Cols2>
MatrixEx<T, _Rows1 + _Type * _Rows2, _Cols1 + (1 - _Type) * _Cols2>
concatenateEx(MatrixEx<T, _Rows1, _Cols1>& mat1, MatrixEx<T, _Rows2, _Cols2>& mat2);
```

- 也可以使用require来进行是行拼接还是列拼接

扩展4

- 模板继承
  - 第一层基类中有数据成员，保存矩阵的数据。这里有两种基类，一种是静态内存分配，一种是动态内存分配。

```
template <typename T>
struct MatrixDynamicData {
  size_t row_;
  size_t col_;
  std::vector<T> elements_;
};

template <typename T, size_t T_row, size_t T_col>
struct MatrixStaticData {
  static constexpr size_t row_=T_row;
  static constexpr size_t col_=T_col;
  std::vector<T> elements_;
}
```

扩展4

- 模板继承
  - 第二层的结构体会根据不同的情况来继承不同的数据基类。一些基础的计算将会在这层进行。

```
template<class T>
struct MatrixBase : public T
{
  template<class T1, class T2>
  void add(const T1& rhs, T2& res)
  {
    for(size_t i=0; i<T::row_*T::col_;++i)
      res.elements_[i]=T::elements_[i]+rhs.elements_[i];
  }

}
```

扩展4

深蓝学院 shenlanxueyuan.com

- 模板继承
  - 第三层的结构为最后的子类。会根据情况来选择继承哪种基类。用特化的方法来实现。

感谢第三期助教-云行月下提供的代码框架和思路

```
template <typename T, size_t... sizes> struct Matrix;

template <typename T>
struct Matrix<T> : public MatrixBase<MatrixDynamicData<T>> {

  Matrix operator+(const Matrix &rhs) {
    Matrix res(this->get_row(), this->get_col());
    this->add(rhs, res);
    return res;
  }
};

template <typename T, size_t T_row, size_t T_col>
struct Matrix<T, T_row, T_col>
    : public MatrixBase<MatrixStaticData<T, T_row, T_col>> {

  Matrix operator+(const Matrix &rhs) {
    Matrix res;
    this->add(rhs, res);
    return res;
  }
};
```

深蓝学院
shenlanxueyuan.com

感谢各位聆听

Thanks for Listening