Assignment 5

本次作业共2道题目。每题设置10个测试案例，其中的5个已经提供用于测试。2道题共计10分。

1. Matrix

题目描述

使用c++的类设计并定义一个`Matrix`（矩阵）类，需要支持以下功能：

- 默认构造函数：初始化了一个元素为0，1行1列的矩阵
- 构造函数：初始化一个所有元素都为零的矩阵，例如：`Matrix m(5, 8);`初始化了一个元素都为0，5行8列的矩阵
- `set`方法：设置第`row`行第`col`列的元素为`value`，例如：`m.set(1, 2, 3);`将第1行第2列的元素设为3
- `get`方法：返回第`row`行第`col`列的元素；若发生`index`越界，则返回`-1`
- `add`方法：实现矩阵加法`m1.add(m2);`
- `mul`矩阵相乘方法：实现两个矩阵相乘`m1.mul(m2);`
- `toString`方法：输出矩阵，行内用空格分隔，行间用换行符分隔 示例：

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

若`m`为以上矩阵，则调用`m.toString();`会输出

```
1 2 3
4 5 6
7 8 9
```

- 重载运算符+：实现矩阵加法`m1 + m2`。
- 重载运算符*：实现矩阵相乘`m1 * m2`
- 重载运算符<<：输出代表矩阵的字符串
- `getErrorCode`方法：返回错误信息。在`Matrix`类中添加一个成员变量`errorCode`，类型为枚举类型`ErrorCode`，其中包含以下错误代码，需检查相关错误，将`errorCode`设为对应错误代码：
  - `NoError`：没有出现错误，为默认值
  - `IndexOutOfRange`：调用`set`或`get`方法时，`index`越界
  - `DimensionMismatch`：进行矩阵运算时，矩阵维度不符合运算要求
  - `InvalidDimension`：调用构造函数时，`rowNum`或`colNum`小于等于0

注意：若运行`Matrix m = m1 + m2;`时，发现`m1`和`m2`的维度不一致，则`m.errorCode`设为`DimensionMismatch`，`m1.errorCode`和`m2.errorCode`仍为`NoError`，且对`m`的其余成员变量的值不做要求

- `printErrorCode`方法：非成员函数，根据错误代码打印错误信息
  - `NoError`："No Error"
  - `IndexOutOfRange`："Index out of Range"
  - `DimensionMismatch`："Dimension Mismatch"
  - `InvalidDimension`："Invalid Dimension"

示例： 若

$$m1 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, m2 = \begin{pmatrix} 5 & 6 \end{pmatrix}$$

则运行

```
Matrix sum = m1 + m2;
printErrorCode(sum.getErrorCode());
```

会输出`Dimension Mismatch`

输入输出格式

输入由不确定行数的指令组成

支持的指令：

- `new Name row col`：创建一个`row`行`col`列，名为`Name`的矩阵
- `set Name row col val`：设置矩阵元素，将名为`Name`的矩阵的第`row`行第`col`列的元素设为`val`
- `get Name row col`：获取矩阵元素并输出，输出名为`Name`的矩阵的第`row`行第`col`列的元素
- `add A B C`：执行`C = A + B`
- `mul A B C`：执行矩阵乘法`C = A * B`
- `print Name`：输出名为`Name`的矩阵
- `error Name`：输出名为`Name`的矩阵的错误信息

提示：

1. 可以用map<string, Matrix>类型存储`Name` → `Matrix`
2. 实现`toString`方法时，可以使用`ostringstream`或`std::to_string`将数值转换成字符串 示例：

```
int num = 10;

ostringstream oss;
oss << num;
oss.str();

std::to_string(num);
```

输入输出示例

输入：

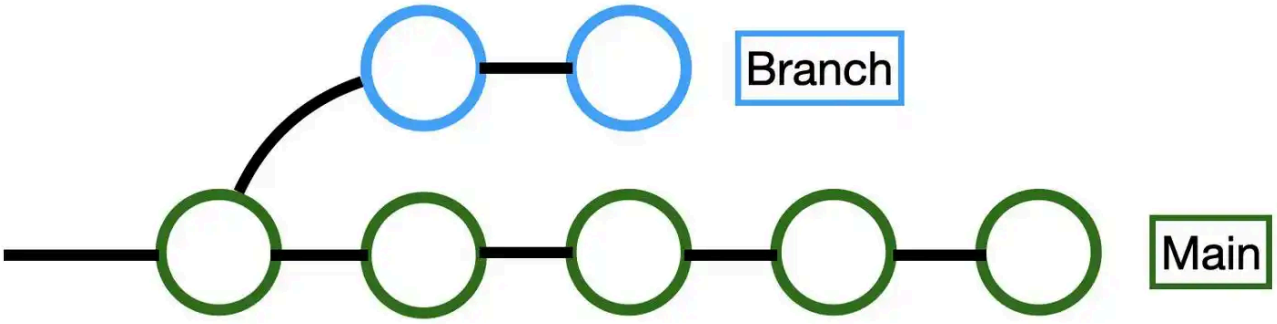
```
new A 2 2
set A 0 0 1
set A 0 1 2
set A 1 0 3
set A 1 1 4
print A
get A 1 1
error A
```

输出：

```
1 2
3 4
4
No Error
```

注意：输出之间需要换行

2. GitArray



背景

在开发大型软件时，版本控制是一个重要概念。例如，在Git这样的版本控制系统中，每次修改文件都会创建一个新的提交（commit），而用户可以自由地回溯到任何一个历史版本。本题的目标是模拟一个支持版本控制的数组，让用户可以方便地管理数组的多个历史版本。

题目描述

请设计一个`GitArray`类，实现一个支持版本管理的数组，元素为`int`类型。数组的每次修改都会自动创建新版本，存储当前所有元素的状态。`GitArray`类存在多个分支，用户可以在分支间切换，也可以查询当前分支任意版本的数组状态，并可以对当前分支进行撤销和恢复操作。

要求实现的方法：

- 构造函数：以数组长度`array_length`为输入，数组初始值均为0，默认分支名为`main`，示例：`GitArray history(5);`数组长度为5 其中，`array_length`是正整数
- `toString`方法：返回当前分支当前版本的数组，形如$(a_1,a_2,.....,a_n)$的字符串，例如调用`history.toString()`应该返回字符串"(0,0,0,0,0)"
- `set`方法：设置当前分支索引为`index`的元素为`value`，并创建新版本（不存在非法输入） 示例：
  - 运行`history.set(1, 3);`后，`history.toString()`应该返回字符串"(0,3,0,0,0)"
  - 运行`history.set(4, 20)`后，`history.toString()`应该返回字符串"(0,3,0,0,20)"
- `get`方法：获取当前分支索引`index`在指定版本`version`的值。其中`version=-1`代表上一个版本，`version=0`代表当前版本，`version=1`代表下一个版本，以此类推。（不存在非法输入） 示例：
  - `history.get(4, 0)`应该返回20
  - `history.get(1, -1)`应该返回3
- `undo`方法：当前分支回退到上一个版本，没有返回值。若没有上一个版本，则输出字符串"No Previous Version to Undo"。 示例：

    - 运行`history.undo()`后，`history.tostring()`应该返回字符串"(0,3,0,0,0)"
    - 第二次运行`history.undo()`后，`history.tostring()`应该返回字符串"(0,0,0,0,0)"
    - 第三次运行`history.undo()`后，输出字符串"No Previous Version to Undo"，`history.tostring()`应该返回字符串"(0,0,0,0,0)"

- `redo`方法：当前分支恢复到下一个版本，没有返回值。若没有下一个版本，则输出字符串"No Next Version to Redo"。 示例：

    - 运行`history.redo()`后，`history.tostring()`应该返回字符串"(0,3,0,0,0)"
    - 第二次运行`history.redo()`后，`history.tostring()`应该返回字符串"(0,3,0,0,20)"
    - 第三次运行`history.redo()`后，输出字符串"No Next Version to Redo"，`history.tostring()`应该返回字符串"(0,3,0,0,20)"

注意：使用`set`方法时，若当前版本后还有其它版本，则在记录新版本前，清空当前版本后的所有版本 例如：若当前`history`的`main`分支从旧到新有版本(0,0,0,0,0), (0,3,0,0,0), (0,3,0,0,20), (0,3,8,0,20), (0,3,8,9,20)，当前版本为(0,3,0,0,20)，若调用`history.set(2, 18);`，则`history`的`main`分支从旧到新有版本(0,0,0,0,0), (0,3,0,0,0), (0,3,0,0,20), (0,3,18,0,20)

- `branch`方法：创建一个以当前版本为基础的新分支，新分支名`branch_name`为参数（分支名不会重复），不同分支拥有独立的版本历史。 示例： 当前在主分支`main`下，调用`history.branch("featureA")`，表示复制`main`分支创建新分支`featureA`，`main`分支的当前版本及其之前的历史版本都存在于`featureA`分支中（`main`分支当前版本之后的历史版本不在`featureA`分支中）。当前分支仍为`main`。

- `checkout`方法：切换到指定分支`branch_name`的最新版本，使其成为当前工作分支。切换后所有操作（如`toString`、`set`、`get`、`undo`、`redo`）都基于该分支。若分支`branch_name`不存在，输出 "Branch not found"，不对分支进行切换。 示例：`history.checkout("featureA")`表示切换到`featureA`分支

输入输出格式

输入的第一行为`length version`，用于初始化数组长度为`length` 接下来由不确定行数的指令组成

支持的指令：

- `print`：输出当前分支当前版本的数组
- `set index value`：设置当前分支索引为`index`的元素为`value`
- `get index version`：获取并输出当前分支索引`index`在指定版本`version`的值
- `undo`：当前分支回退到上一个版本
- `redo`：当前分支恢复到下一个版本
- `branch branch_name`：创建新分支`branch_name`
- `checkout branch_name`：切换到指定分支`branch_name`的最新版本

输入输出示例

输入：

```
5
set 1 3
set 4 20
undo
print
undo
print
undo
print
```

输出：

```
(0,3,0,0,0)
(0,0,0,0,0)
No Previous Version to Undo
(0,0,0,0,0)
```

提交格式

在编写完全部程序并测试完毕后，**将2个文件夹中除要求以外的文件删除之后一起打包压缩**，最后用自己的学号命名，即可提交。 你提交的文件结构应该如下（可以包含额外的头文件，如utilities.h用来实现一些容器的辅助功能）：

```
<your student number>.zip
|- 1_matrix
|  |- main.cpp
|  |- matrix.h
|  |- matrix.cpp
|
|- 2_gitArray
|  |- main.cpp
|  |- gitArray.h
|  |- gitArray.cpp
```