Lab 3: 汉诺塔

在Lab 3中，你需要设计并实现一个游戏：汉诺塔。

1. 汉诺塔问题

汉诺塔是一个著名的数学问题。它由三根杆子和若干不同大小的盘子组成。开始时，所有的盘子都在第一根杆子上，并按照从上到下大小升序排列（也就是说，最小的在最上面）。这个问题的目标是将所有盘子移到另一根杆子上，并遵守以下简单的规则：

1. 每次只能移动一个盘子。
2. 每次移动都是将其中一根杆子的最上面的盘子取出，放到另一根杆子上。
3. 任何较大的盘子都不能放在较小的盘子上面。

解决这个问题的经典方法是递归。该算法可以描述为以下伪代码：

```
function hanoi(n, A, B, C) {  // move n disks from rod A to rod B, use rod C as a buffer
     hanoi(n - 1, A, C, B);
     move(n, A, B);  // move the nth disk from rod A to rod B
     haoni(n - 1, C, B, A);
}
```

2. 实验描述

在本实验中，杆子的数量总是等于3，盘子的数量可以是1～5。其中，第1根杆子为初始杆，第2根为目标杆。

本实验的任务包括以下内容：

- 完成一个交互式的汉诺塔游戏程序，根据用户输入的指令移动相应的盘子，并在用户胜利时打印提示；
- 通过命令行界面，将汉诺塔游戏的状态绘制出来，包括3根杆子和若干盘子；
- 根据汉诺塔问题的递归算法，提供一个自动求解程序，能够从任一状态出发，通过若干步移动达到目标状态。

具体而言，程序的流程如下：

1. 首先，程序打印`How many disks do you want? (1 ~ 5)`，要求输入盘子的数量（要求为1～5）。如果输入`Q`，则退出程序。不合法的输入应当忽略。
2. 接下来程序将打印汉诺塔的状态，随后打印`Move a disk. Format: x y`，要求用户给出指令。指令的形式是`from to`（例如，`2 3`的意思是将杆2最上面的盘子移动到杆3上），不合法的输入或是不可执行的指令应该忽略。在这之后，无论指令是否合法，程序总是重新打印一遍当前状态，并重新要求用户输入。
3. 如果输入的指令为`0 0`，则进入自动模式。程序需要首先将用户已经执行的指令反过来执行一遍，复原到初始状态，然后再按照递归算法执行。每次执行时，程序通过输出`Auto moving:x->y`告知用户所执行的指令。**注意：即使有其他方法从当前状态直接到达目标状态，也请按照先复原后执行的方式进行。这是因为自动评测的时候会直接比对输出内容。**
4. 无论是通过用户指令或是自动模式，只要达成目标状态（即所有盘子都移到杆2上），就打印游戏胜利的提示信息，然后重新回到第1步。

打印汉诺塔状态的要求：我们用`|`表示杆子，`-`表示底座，一排`*`表示盘子。每个盘子从小到大分别用3、5、7、9、11个`*`表示（最多5个盘子）。**无论盘子数量多少，输出的整个画布的大小固定为11x41。**

例如，一共5个盘子，均按照从小到大放在杆1上，此时输出的结果应该如下：

```
     |             |             |
    ***            |             |
     |             |             |
   *****           |             |
     |             |             |
  *******          |             |
     |             |             |
 *********         |             |
     |             |             |
***********        |             |
-----|-------------|-------------|-----
```

而如果只有3个盘子，输出如下（注意画布的大小不变）：

```
     |             |             |
     |             |             |
     |             |             |
     |             |             |
     |             |             |
    ***            |             |
     |             |             |
   *****           |             |
     |             |             |
  *******          |             |
-----|-------------|-------------|-----
```

3. 样例输入输出

这里给出两个样例，分别使用自动模式和手动模式。这里行首的`>` `<`代表是程序的输入还是输出。

样例1：

```
< How many disks do you want? (1 ~ 5)
> 2
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<     ***            |             |
<      |             |             |
<    *****           |             |
< -----|-------------|-------------|-----
```

```
< Move a disk. Format: x y
> 0 0
< Auto moving:1->3
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<    *****            |             ***
< -----|--------------|--------------|-----
< Auto moving:1->2
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |            *****           ***
< -----|--------------|--------------|-----
< Auto moving:3->2
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |             ***             |
<      |              |              |
<      |            *****           |
< -----|--------------|--------------|-----
< Congratulations! You win!
< How many disks do you want? (1 ~ 5)
> Q
```

样例2：

```
< How many disks do you want? (1 ~ 5)
> 2
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
<      |              |              |
```

```
<      |             |             |
<      |             |             |
<     ***            |             |
<      |             |             |
<    *****           |             |
< -----|-------------|-------------|-----
< Move a disk. Format: x y
> 1 2
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<    *****          ***            |
< -----|-------------|-------------|-----
< Move a disk. Format: x y
> 2 3
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<    *****           |            ***
< -----|-------------|-------------|-----
< Move a disk. Format: x y
> 1 2
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |            *****          ***
< -----|-------------|-------------|-----
< Move a disk. Format: x y
> 3 2
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
<      |             |             |
```

```
<      |             |             |
<      |             |             |
<      |            ***            |
<      |             |             |
<      |           *****           |
< -----|-------------|-------------|-----
< Congratulations! You win!
< How many disks do you want? (1 ~ 5)
> Q
```

4. 面向对象和数据结构

要完成这个实验，你需要了解C++面向对象编程以及基本的数据结构知识（如栈和队列）。
在本实验中，你需要根据我们的设计来实现对应的数据结构。

4.1 UniquePtr

本实验中，实现的栈和队列是由链表的方式实现的，并且链表节点`node.h`已经为你实现好了。与你常见的实现不同，我们为你实现的链表有自己特殊的属性——所有权机制。所有权是程序设计中一种常见的设计方法，表达**某个数据属于某个变量**的概念。在我们实现的链表节点中，**一块数据只能被一个所有者拥有**，即数据是独占的。下面是一个简单的例子：

```
int *p = new int(2);  // p申请了一块新内存
int *p2 = p;          // p2复制p的值，也指向同一块内存
*p += 1;
assert(*p2, 3);       // p的修改会影响到p2
```

上面这个例子中，`p`和`p2`两个指针都指向同一个内存块，即`p`和`p2`共享了内存块的所有权。下面我们使用封装指针的类型——`UniquePtr`，来演示如何使用独占所有权：

```
// p是指向内存块的指针，内存块的值为1
UniquePtr<int> p = MakeUnique<int>(1);

// p2也想要指向指针p所指向的内存块，引发编译错误
UniquePtr<int> p2 = p; // compile error

// p将自己所有的内存块移动给p3，p变成nullptr，p3成为内存块的所有者
UniquePtr<int> p3 = std::move(p);

assert(p, nullptr);
assert(*p3, 1);
```

除了独占所有权这一特性之外，`UniquePtr`还使用了C++的RAII来保障内存安全。Resource acquisition is initialization (RAII)是C++的特性，通过构造函数和析构函数的配套使用，`UniquePtr`可以在构造函数中申请内存，在析构函数中释放内存，从而避免手动的`new`和`delete`。

```
{
  UniquePtr<int> p = MakeUnique<int>(1);
  fn(p); // 对p及指向的内存进行操作
}
// p的生命周期结束，内存自动被释放
```

在实验中，我们为你定义好了`UniquePtr`的原型，你需要完成具体的实现。请注意，实现`UniquePtr`很容易产生错误，包括意外的空指针、不经意的内存泄漏。实验中有一个使用googletest编写的测试框架（`lab3_test.cpp`），我们会通过详细的单元测试来确保你的`UniquePtr`是正确的。我们也鼓励你自己对`UniquePtr`进行测试，具体测试的编写方式可以参考`lab3_test.cpp`。

4.2 链表、栈、队列

实验在`UniquePtr`的基础上，构建其他的数据结构。我们为你定义好了`Stack`和`Queue`的原型，你需要完成具体的实现。

请注意，你只需要构建**单向链表**的栈和队列。每个数据结构都分为`xxx.h`和`xxx_impl.h`，`xxx.h`是我们为你实现好的原型，你需要实现的是`xxx_impl.h`里的逻辑。

在其他的地方使用这些数据结构，只需要包含`xxx.h`即可。

5. 提示

我们已经准备了本实验的代码框架。其中每个文件的用途说明如下：

- `hanoi.cpp`中含有main函数，请从这里开始编写程序的整体逻辑。
- `board.cpp`和`board.h`中实现了Board类，这个类用来表示汉诺塔游戏的状态。Board类主要的成员函数分别解释如下：
  - `move`函数用于执行移动指令，其中第三个参数`log`代表该操作是否为用户手动执行，如果是则需要记录至历史；
  - `win`函数用于判断游戏是否胜利；
  - `draw`函数向控制台打印当前状态（按照上面所说的格式）；
  - `autoplay`函数可以开始自动模式。
- `canvas.h`是提供的画布工具。该文件实现了Canvas类，该类可以创建一个11x41的缓冲区，允许你先在这个缓冲区上绘画，然后再一起输出。你可以在`Board::draw`函数中使用该Canvas类。
- `rod.h`和`rod.cpp`是提供的工具类，其中包含了修改柱子的逻辑代码，通过与Canvas类交互起到修改画布信息的作用。
- `disk.h`和`disk.cpp`是提供的工具类，其中包含了修改盘子的逻辑代码，通过与Cancas类交互起到修改画布信息的作用。
- `queue.h` `stack.h`分别实现栈和队列，他们都使用了`node.h`和`unique_ptr.h`的逻辑。这些文件是接口定义和我们为你提供好的一部分功能实现，每个容器都对应有`xxx_impl.h`的头文件，内部包含了真正的函数实现，需要你来完成。

请你按照前面对程序功能的描述，补全代码中的TODO注释，实现汉诺塔游戏程序。

你需要注意以下事情：

1. 请不要在代码中使用STL容器和智能指针。禁止的容器类包括：`std::vector` `std::stack` `std::queue` `std::list` `std::unique_ptr`，其他类（如`std::pair`）的使用不受影响。不使用STL容器会在评分中占10分。该项分数会通过评测程序自动检查，如果评分有误请及时联系负责的助教进行人工检查。
2. 你需要**完全按照给出的代码框架来编写**，并且只修改需要你修改的文件，提交的压缩包中只包含你所完成的文件。请注意，测试时，不需要你修改的文件会被我们用原始文件覆盖，如果你也修改了这些文件，最终可能会导致平台无法编译你的代码。
3. `UniquePtr`只能通过`std::move`来移动所有权，不能直接拷贝赋值/构造。你的实现不需要考虑如何做到这一点，只需要编写移动赋值和移动构造函数即可。
4. 你可以调用`MakeUnique`得到一个任意类型T的`UniquePtr`，直接传入构造T所需要的参数即可。下面是一个更详细的例子：

```
struct A {
  int a;
  int b;
  A(int a, int b): a(a), b(b) {}
}

UniquePtr<A> p = MakeUnique<A>(1, 2); // a = 1, b = 2
```

8. `UniquePtr`的定义中包含了`reset`和`release`函数，注释中描述了这两个函数具体的语义。只使用`reset`和`release`的简单组合就可以实现移动构造和移动赋值，同时我们也会测试你的`reset`和`release`是否正确实现。
9. 在实现堆和栈的时候，多使用`assert`来检查指针是否为`nullptr`是一个很好的习惯，能够帮助你节省debug的时间。
10. 在实现自动模式时，请务必先将用户的所有操作一一复原，然后再按照标准的递归流程进行。
11. 遇到不合法的输入时，请直接忽略，重新请求用户输入。如果是在游戏进行时遇到不合法的输入，要当作执行一次无效操作，先输出一遍游戏状态，再请求用户输入。测试用例中会有1个用于测试程序面对非法输入的行为。不合法的输入可能包括：

- 需要输入数字时输入了其他字符；
- 输入的数值过小或过大；
- `from`是空杆；
- 试图将大盘放到小盘上；
- 等等。

12. 你只需要在栈、队列的实现中使用`UniquePtr`，在表达数组的时候，可以直接使用裸指针（我们为你提供的`board`、`canvas`、`rod`、`disk`也是使用裸指针的）。
13. 注意提交的应当是一个7z压缩包，且压缩包内首先有一个`lab3`文件夹，文件夹内再包含源代码，错误的文件结构会导致测试失败。
14. 如果你在编译的时候遇到提示网络问题，可能是由于依赖中会拉取`googletest`的源代码，而你的网络无法访问github导致的。你可以将`CMakeList.txt`的googletest网址改成gitee的镜像源：

```
FetchContent_Declare(
        googletest
        # 指定googletest的仓库URL
-       GIT_REPOSITORY https://github.com/google/googletest.git
+       GIT_REPOSITORY https://gitee.com/mirrors/googletest.git
        # 你可以指定一个特定的提交、分支或标签
        GIT_TAG v1.14.x
)
```

5. 提交和评分

你需要提交的文件包括：

```
hanoi.cpp
board.cpp
rod.cpp
queue_impl.h
stack_impl.h
unique_ptr_impl.h
```

请将你的源代码打包成`lab3-XXX.7z`（其中`XXX`是你的学号），然后上传到canvas上。7z文件的文件夹结构应当如下：

```
lab3-XXX.7z
   |--- lab3
          |--- xxx.h
          |--- xxx.cpp
          |--- ...
```

评分标准：编译通过30分，5个测试用例每个10分，不使用STL容器和智能指针10分，`UniquePtr`的单元测试10分，总分100分。