图形界面

题目内容

刷新

我们来尝试模拟一个图形界面：在一个窗口(window)内，包含若干控件(widget)，例如按钮(button)、文本框等等。所有的控件都由Widget基类派生得到。

每一个控件都属于一个窗口，因此 Widget 有一个 `Window*` 成员 `_parent` 用来记录其所属的窗口（当控件暂不属于任何窗口时 `_parent` 为 `nullptr`）。

为方便定位，同一窗口内的每一个控件都有一个独一无二的名字。我们可以这样定义Widget基类：

```
class Widget {
    string _name;
    Window* _parent;

public:
    Widget(string name) :_name(name),_parent(nullptr) {}
    string getName() const { return _name; }
    Window* getParent() const { return _parent; }
    void setParent(Window* parent) { _parent = parent; } // 将 Widget对象 与 Window对象 绑定
    virtual ~Widget() {}
};
```

定义好了Widget基类，我们就可以定义Window类了。一个窗口包含若干个控件，可以用 `list<shared_ptr<Widget>>` 保存；我们还希望能够通过控件的名字获取指向控件的指针和引用，Window类有 `getPointerByName` 和 `getElementByName` 两个成员函数，通过一个name字符串获取对应的widget；最后，Window类还可以动态添加控件，通过 `addElement` 方法，可以将widget添加到Window中。

```
class Window {
    list<shared_ptr<Widget>> Widgets;

public:
    // 返回一个指向名称为name的控件的指针
    shared_ptr<Widget> getPointerByName(string name);

     // 返回一个指向名称为name的控件的引用
    Widget& getElementByName(string name);

    // 将控件w添加到窗口中。如果w的名字已经存在，则不添加w，返回false；否则返回true。
    bool addElement(shared_ptr<Widget> w);
};
```

```
// 使用Window类的例子
// 这里假设Button（按钮）类是Widget的派生类
Window win; // 创建一个空窗口，没有控件
win.addElement(make_shared<Button>("btn1")); // 向窗口win中添加一个按钮控件，名为btn1，返回true
win.addElement(make_shared<Button>("btn1")); // 这条语句会返回false，不会添加任何控件
cout << win.getPointerByName("btn1")->getName() << endl; // 输出 btn1
cout << win.getElementByName("btn1").getName() << endl; // 输出 btn1
```

1

我们希望，对于一个具体的控件，例如按钮，可以动态设置其被单击或双击后的反应。 你需要实现两个基类：Click类和DoubleClick类，使得派生自这两个类的控件分别能够响应单击和双击事件。例如，按钮可以响应单击和双击，通过下面的代码就可以实现：

```
class Button :public Widget, public Click, public DoubleClick {
public:
    Button(string name) :Widget(name) {}
};

Button b("btn1");
b.setClickCommand([]{ cout << "btn1 was clicked!"; }); // 此处展示的 `[]{ cout << "btn1 was clicked!"; }` 是lambda表达式
b.setDoubleClickCommand([]{ cout << "btn1 was double-clicked!"; });
b.click();  // 输出 btn1 was clicked!
b.doubleClick();  // 输出 btn1 was double-clicked!
```

`setClickCommand` 接受一个void()的可调用对象，设置单击时执行的代码（每次覆盖掉之前的设置）； `click()` 则运行之前设置的代码（如果控件之前从未执行 `setClickCommand` ，则 `click()` 什么也不做）。 `setDoubleClickCommand` 和 `doubleClick()` 同理。

2

考虑另一种控件——多选框(checkbox)。一个多选框有两种状态：选中和未选中，默认是未选中状态。多选框可以响应单击：单击时，默认改变多选框的选中状态（选中改为未选中，未选中改为选中）

```
class Checkbox: public Widget, public Click {
    // TODO ...
}
Checkbox c1("c1"); // c1未选中
c1.click(); // c1选中
c1.click(); // c1未选中
```

3

我们来尝试为按钮增加一些功能。 首先是多选框的全选和清空功能。你需要编写 `SelectAll` 类和 `ClearAll` 类，实现下面的效果：

```
Window win;
// 向窗口中添加1个按钮和2个多选框
auto btn1=make_shared<Button>("btn1");
auto c1=make_shared<Checkbox>("c1");
auto c2=make_shared<Checkbox>("c2");
win.addElement(btn1); // 添加按钮
win.addElement(c1); // 添加多选框c1
win.addElement(c2); // 添加多选框c2

list<shared_ptr<Checkbox>> l = {c1,c2}; // 设置全选/清空的是哪几个多选框
btn1->setClickCommand(SelectAll(l)); // 将btn1的单击指令设置为将c1、c2全选
btn1->click(); // c1、c2均选中
btn1->setClickCommand(ClearAll(l)); // 将btn1的单击指令设置为将c1、c2清空
btn1->click(); // c1、c2均未选中
```

你还需要编写 `Submit` 类， `Submit` 类将多选框的选择情况输出：

```
Window win;
// 向窗口中添加1个按钮和2个多选框
auto btn1=make_shared<Button>("btn1");
auto c1=make_shared<Checkbox>("c1");
auto c2=make_shared<Checkbox>("c2");
win.addElement(btn1); // 添加按钮
win.addElement(c1); // 添加多选框c1
win.addElement(c2); // 添加多选框c2

list<shared_ptr<Checkbox>> l = {c1,c2}; // 设置输出的是哪几个多选框
btn1->setClickCommand(Submit(l)); // 将btn1的单击指令设置为将c1、c2选择情况输出
btn1->click(); // 输出：c1 is not selected\nc2 is not selected\n
c1->click(); // 单击c1
btn1->click(); // 输出：c1 is selected\nc2 is not selected\n
```

4

最后，我们为窗口添加撤销功能。

以下几个操作会改变多选框选中情况：

- 单击多选框，即 多选框的 `click` 操作
- 全选多选框，即 `SelectAll`
- 清空多选框，即 `ClearAll`

你需要在Window类中添加一个成员函数 `bool undo()` 实现撤销功能，即，每次执行 `undo()` 成员函数，撤回前一次对该窗口中的多选框选择情况的修改。注意：无论全选/清空多选框是否实际上修改多选框的选中情况，都算作一次撤销操作。

如果没有可供撤销的指令，返回false；否则撤销并返回true。

例1：

```
Window win;
// 向窗口中添加1个按钮和2个多选框
auto btn1=make_shared<Button>("btn1");
auto c1=make_shared<Checkbox>("c1");
auto c2=make_shared<Checkbox>("c2");
win.addElement(btn1); // 添加按钮
win.addElement(c1); // 添加多选框c1
win.addElement(c2); // 添加多选框c2
list<shared_ptr<Checkbox>> l = {c1,c2};
btn1->setClickCommand(ClearAll(l));

c1->click(); // c1选中，c2未选中
win.undo(); // c1未选中，c2未选中，返回true
win.undo(); // 返回false，因为已经到头了，无法继续撤销
btn1->click(); // c1未选中，c2未选中
win.undo(); // c1未选中，c2未选中，返回true（即使清空没有改变c1,c2的选中情况，也可以撤销，此时撤销相当于什么都没做）
```

我们将提供 `Makefile`，`main.cpp`， `command.h`， `event.h`，和 `window.h`，下载地址：点此下载 (/staticdata/1812.yweD4BGoULF64KUv.pub/CVKAZcXdwvV2xsK8.download.zip/download.zip)

输入格式

第一行是一个整数 n 后 n 行，每一行是一条指令。指令总共有4种：

- `create button/checkbox AA`，指在窗口中添加一个名称为AA的按钮/多选框。如果名称与前面创建的控件重复，输出 `AA already exists.`；否则输出 `AA has been created successfully.`；**其他指令只会在 `create` 指令之后出现**
- `setclick/setdoubleclick AA clear/select/submit m A B C ...`，将名称为AA的按钮单击/双击时的操作设置为清空/全选/提交，数字 m 表示与按钮关联的多选框个数，后面紧接着 m 个多选框名称；**输入数据保证 按钮和多选框 均已通过 `create` 指令创建**
- `click/doubleclick XX`，单击/双击XX；**输入数据保证 按钮 已通过 `create` 指令创建**
- `undo`，撤销一次。如果不能撤销，输出 `can't undo.`

输入样例

```
18
create button b1
create button b1
create button b2
create checkbox c1
create checkbox c2
create checkbox c3
setclick b1 clear 2 c1 c2
setdoubleclick b1 select 2 c2 c3
setclick b2 submit 3 c1 c2 c3
undo
doubleclick b1
click b2
click c1
click b2
click b1
click b2
undo
click b2
```

样例输出：

```
b1 has been created successfully.
b1 already exists.
b2 has been created successfully.
c1 has been created successfully.
c2 has been created successfully.
c3 has been created successfully.
can't undo
c1 is not selected
c2 is selected
c3 is selected
c1 is selected
c2 is selected
c3 is selected
c1 is not selected
c2 is not selected
c3 is selected
c1 is selected
c2 is selected
c3 is selected
```

数据范围

$1 < n \leq 200$

$m \geq 1$

共10个测试点，第1个点为输入样例，测试点 2-6 保证不含有 undo 操作。

评分规则

每个测试点都为10分，即1-3小问共60分，第4小问40分。

提交格式

你需要提交多个文件，包含 `window.h`、 `command.h`、 `event.h`，可以不包括提供的 `main.cpp`。

你应该将你的文件打包成一个zip压缩包并上传。注意：你的文件应该在压缩包的根目录下，而不是压缩包的一个子文件夹下。评测时，OJ会将 `main.cpp` 贴入你的目录下进行编译并执行。

语言和编译选项

#	名称	编译器	额外参数	代码长度限制
0	custom	make		65536 B

递交历史

#	状态	时间

递交答案

语言和编译选项 custom

```
1
```

提交

文件请拖入编辑器中，或 上传文件