# Dynamic Tree 个人报告

**连佳宜　3170104169**

## 一、　分工任务

### 1、VIEW 模块界面设计

1.1　数据结构选择框

1.2　数据输入框与插入、删除、查找、遍历按键

1.3　主界面，节点不同状态显示

1.4　添加节点点击删除功能

### 2、APP 模块整理合并

2.1 view model 模块关联 model 模块

2.2 view model 模块与 view 模块双向关联

2.3 程序运行

## 二、　设计思路

### 1、VIEW 模块界面设计

使用 QT 库进行界面设计。调用 QT 库中的按键、选择框、数字输入框等部件。

对于主界面，显示树结构六层节点。使用布局部件确定每个节点及其箭头的位置，初始化时将其隐藏。当得到显示节点的信号时重新显示。

将按键绑定在函数上，使用同一个函数基类构建函数并在需要时传递参数。设置 set 函数等待与 view model 模块的函数绑定。

在节点上绑定点击删除函数。若收到点击，读取当前节点的属性传递参数，同时调用删除按键绑定的函数

添加信号槽接受 view model 传递的信号。分为状态修改信号与操作失败信号：

- 状态修改信号主要为改变节点状态与显示旋转标志两种。

  在收到节点改变信号时，根据得到节点的坐标重新绘制节点：更新节点上的信息，根据节点状态用不同颜色绘制或隐藏节点。

  当收到显示旋转标志时，根据传递的坐标与旋转类型显示旋转部件。
- 操作失败信号在查找与删除不存在的节点时弹出未找到节点的提示窗口。

## 2、APP 模块设计

新建 view 模块、view model 模块、model 模块

将 model 模块整体绑定至 view model 模块中。

将 view 模块中的节点属性与 view model 绑定，指针指向同一个地址使 view model 改变节点属性使 view 可以直接获得更新值。

将 view 中按键函数与 view model 中对于的函数绑定。

将 view 模块的信号槽与 view model 模块绑定。

# 三、 模块说明

## 1、VIEW 模块设计

### 1.1 按键函数设计

插入按键举例。

在 app 模块中通过传入的函数指针初始化绑定 view model 中的函数。

将插入按键绑定 InsertAction 函数，点击时获取数字输入框中的值传递参数，同时执行函数。

```
1. void ViewClass::InsertAction()
2. {
3.     std::any param(std::make_any<int>());
4.     param = spinBox->value();
5.     m_cmdInsert->SetParameter(param);
6.     m_cmdInsert->Exec();
7. }
8.
9. void ViewClass::set_InsertCommand(const std::shared_ptr
10.                                     <ICommandBase>& cmd) throw()
11.{
12.    m_cmdInsert = cmd;
13.}
```

### 1.2 状态修改信号显示设计

分为节点修改信号与旋转标记信号，当捕捉到 view model 模块转递的信号时根据传入的字符串调用不同的函数。函数中根据同时设置的 node 属性调整页面显示。

```
1.  void MainWindowPropertySink::OnPropertyChanged(const std::string& str)
2. {
3.     if( str == "Change_Node" ) {
4.         m_pW->changeNodeFunction();
5.     }
6.     else if( str == "Rotate_Node" ) {
7.         m_pW->RotateFunction();
8.     }
9. }
```

### 1.3 操作失败信号显示设计

这里不考虑插入失败的情况。在查找或删除操作中若不存在目标节点，则返回操作失败。捕捉到操作失败时，弹出提示窗口。

```
1.  void MainWindowCommandSink::OnCommandComplete(const std::string&
    str, bool bOK)
2. {
3.     if( str == "Insert" );
4.     else if( str == "Find" || str == "Delete" )
5.     {
6.         if( !bOK )
7.         {
8.             QMessageBox box;
9.             box.setWindowTitle("Find Error");
10.            box.setText("The value has not been found in the curr
   ent tree.");
11.            int res = box.exec();
12.        }
13.    }
14.}
```

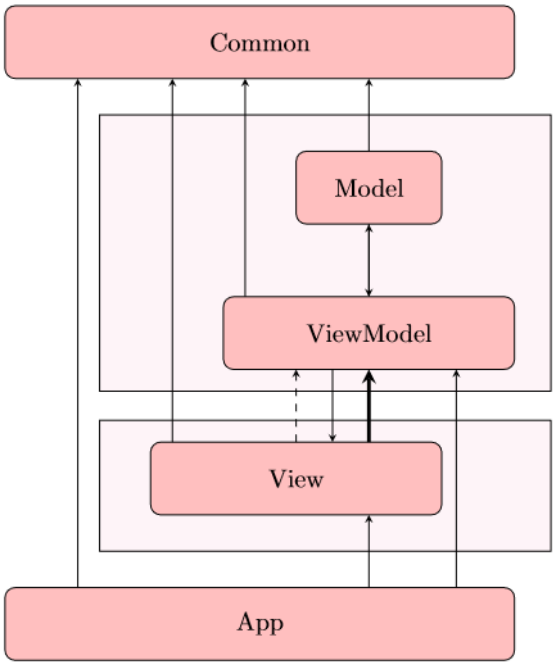
## 2、APP 模块设计

根据 MVVM 框架，通过函数指针传递将模块绑定。

View 和 view model 模块之间无需关联。通过基类派生实现模块完全分类。

对于 view model 和 model，通过 view model 的设置函数直接绑定 model 模块。

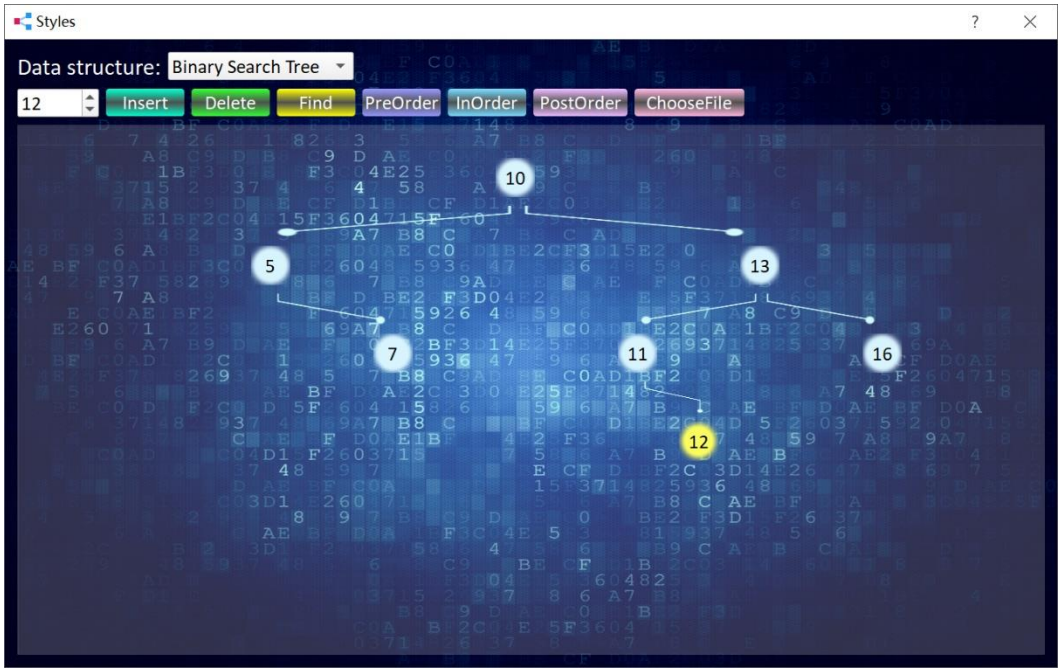
对于 view 和 view model，分别设置节点的共有参数，view model 传至 view 的信号槽，view 绑定 view model 的函数等，实现两者的关联。

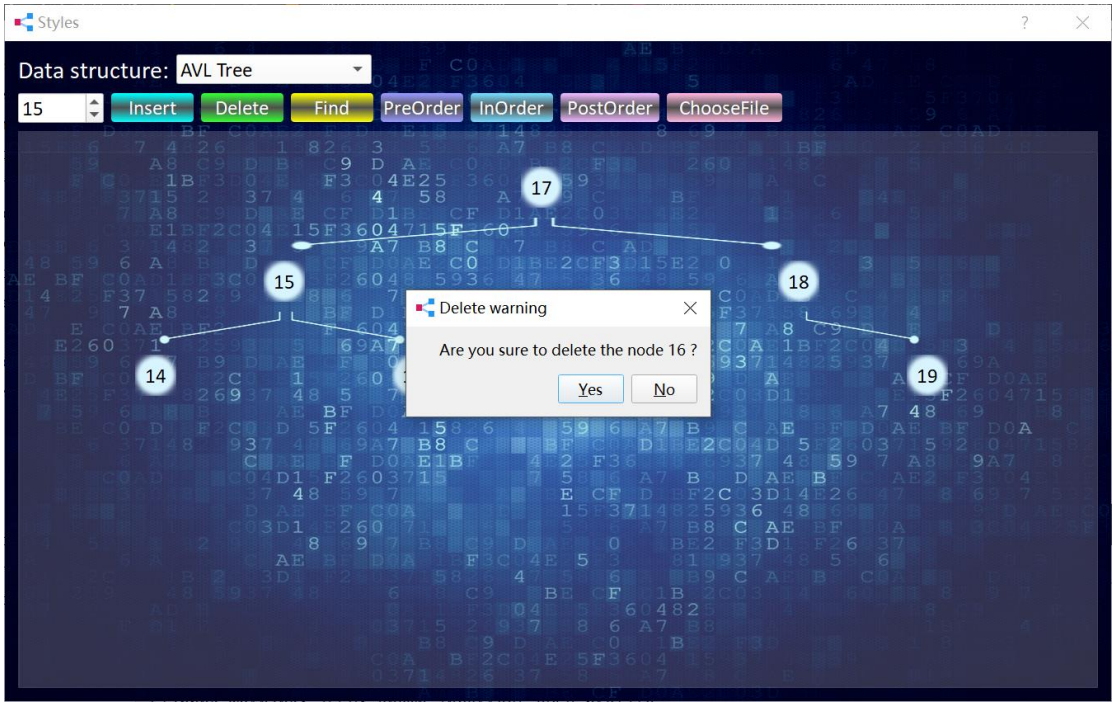



模块关联示意图

在 common 中，定义公有的基类、参数结构、辅助函数等。

## 四、 运行效果图






## 五、 心得体会

课程首先向我们介绍了在应用型程序中工具链的使用，主要为版本控制与持续集成的使用，同时接触了单元测试、代码覆盖率测试、编译工具等。

在课程项目开发中，通过版本控制和持续集成进行小组内的合作，体会到了版本控制在团队合作中的便捷与高效。持续集成在小型程序中作用较小，但实时的集成编译和发布可以使版本有效的更新迭代。

通过课程的学习，可以成功的配置使用持续集成应用，对版本控制器的使用也逐渐熟练高效，成功掌握了新的技能。

课程的重点是程序框架的介绍，分为 MVC，MVP，MVVM 等框架。在之前的团队合作程序中，对程序框架并没有认真的搭建，很容易出现协作困难的情况。在学习了不同的框架后开始了解合理的程序框架的重要性。

在课程项目开发中，我们首先使用了 MVC 框架，MVC 框架在小型程序中直观，但不同模块直接相互耦合，导致后期的开发效率逐渐降低。在感受到 MVC 框架的不足后，我们决定尝试更换为 MVVM 框架。最终搭建出 MVVM 框架重新进行功能的搬迁修改，深刻的体会到 MVVM 框架的优越性。不同模块可以独立运行测试，最后通过 APP 层合并，使多人合作效率明显提高。

通过示例程序的代码学习，对 MVVM 框架有了深入的了解和学习，也体会到了工业化程序与学校课程中程序的不同。

课程的另一个体会是工业要求的重要性，无论是工具链还是程序框架，我们的目标更主要的是大型程序，此时代码更着重于便于维护于协作。在验收时老师提出的数据结构工业上红黑树的使用也体现出了学校中程序于工业上的不同。

## 六、 改进建议

小学期学习到了在其他课程中难以接触到的方向，使我们对工业化程序的设计有了更好的认识，收获颇丰。

这里提一个小小的建议，希望老师能对 MVVM 框架具体的实现方式有更好的介绍，可以通过对代码进行分析，不同基类的作用功能、派生使用介绍等方式，使同学对 MVVM 框架中信号、指令等的传递过程有更好的认识，便于在程序一开始就进行框架的搭建和使用。

谢谢老师！