说明  
代码结构在各个.h中有更具体的表现 （注释采用GBK编码）  
  
整体框架：  
图 Graph  
一个对象对应一张计算图的所有操作。  
变量的输入void creat\_nodes\_1();  
节点的输入void creat\_nodes\_1();  
用户输入操作void commands();  
map<string, Data\_Node\*>： item，存储计算图中用户会访问的所有数据结点（使用变量名访问），abandoned

存储重名后用户不再访问的数据结点（由图中指向它的节点访问）。  
map<int, float> answers存储各次操作的结果。  
~Graph()负责整张图的析构，所有new的数据节点、操作符节点的内存释放。（操作符节点由后方数据节点析构）  
函数crossroad识别输入的不同种类运算符，并将相应参数传递给4个initialize\_operator\_XXX函数，对应1元、2元、3元（逻辑运算符）和COND运算符的对象建立。  
函数string\_to\_int实现读取的string到数据int类型的转换。  
计算图结构：由各个节点存储的数值和前方节点的指针形成结构（以运算顺序在先的为前方）。  
  
所有的节点（见图）：  
抽象类 节点 Basic\_Node，  
图中所有节点的框架,全都使用public继承。使用Basic\_Node\*传递所有节点（即数据节点Data\_Node和运算符节点Operation\_Node ）完成功能。  
数据成员value为节点储存的值，protected属性（如Const数据的value不可修改），读取使用get\_value()，更改数值使用set\_value()。  
bool assigned为记录当前节点是否被计算过，用于避免重复计算，protected属性， 使用clear\_buffer()在用户每次操作之后将assigned状态还原。  
Basic\_Node\* EVAL()为具体功能实现，从EVAL要求的变量开始，递归访问前方指向的节点，判断已计算过（assigned == true）、前方已经报过错不再继续（return nullptr）或进行计算。  
string get\_name()、string get\_type()为数据成员设置属性的接口。  
  
计算符基类 Operation\_Node  
新增protected:  
 vector<Basic\_Node\*> prev\_Datas; //记录前方的数据节点  
各个运算符子类Operation\_XXX 只有构造函数和EVAL()具体实现不同。  
  
数据基类 Data\_Node  
新增protected:  
 string name;  
 string type;  
 Basic\_Node\* prev\_Operation; //记录前方一个运算符节点

各子类中，除type不同外的区别：  
Constant\_Node中 set\_value函数禁用  
Placeholder\_Node中EVAL函数对Placeholder missing报错。