# MaP 设计说明书

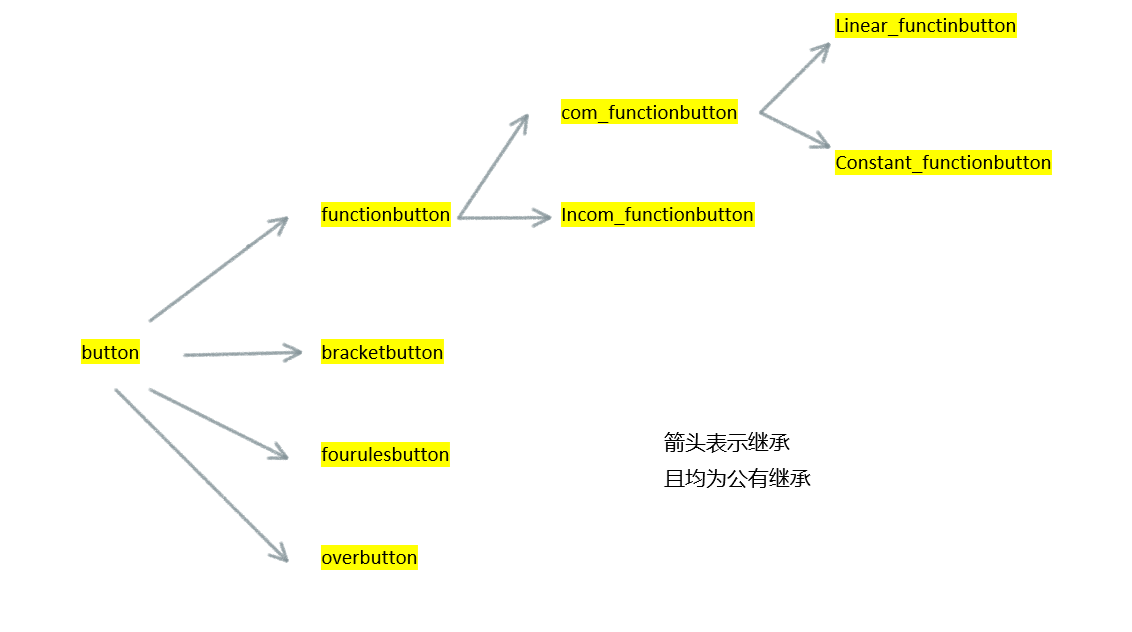
秦跃 机械00 2020010273

## 类设计图

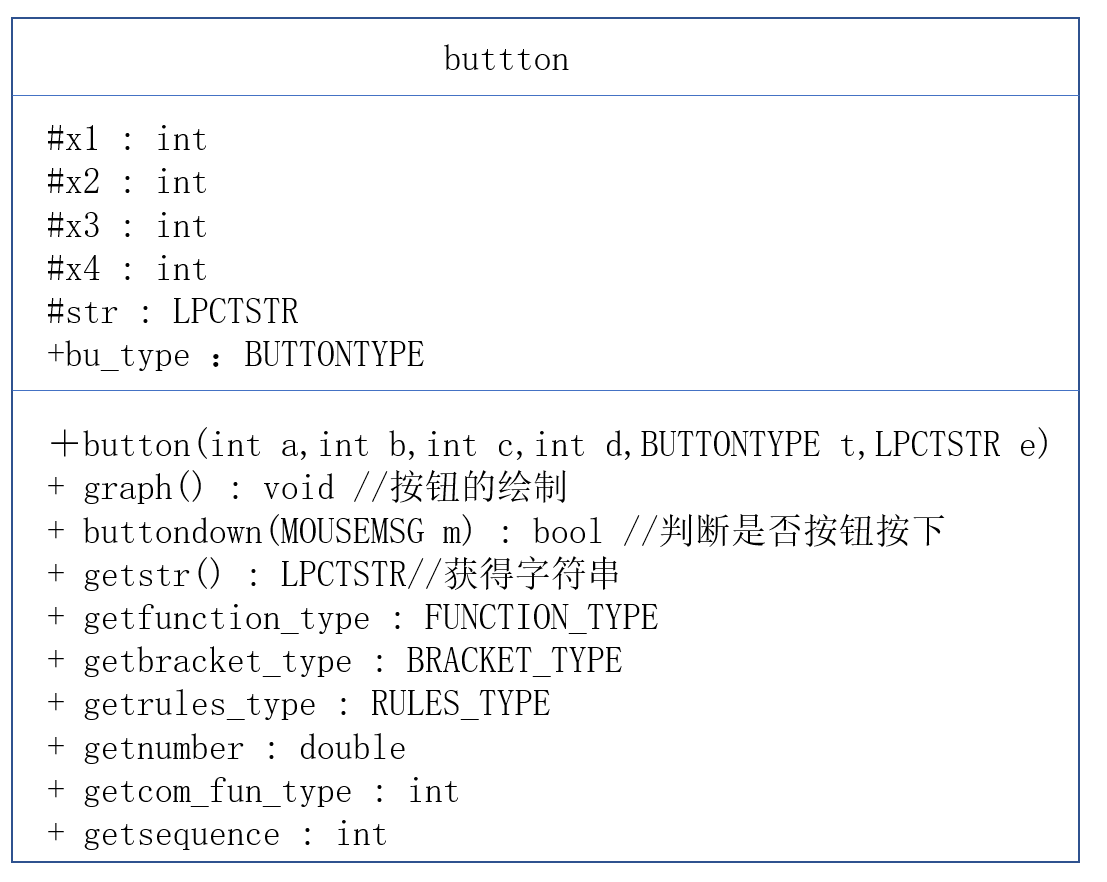
### button类

button类的设计是为了刻画输入函数界面上的这些代表一定含义的按钮，既要储存按钮本身的大小等性质，又要储存它背后的数学含义，故设计了以下类：

#### 1.1.1、继承关系图



#### 1.1.2、成员函数与成员变量说明



button基类的保护成员变量int x1,x2,y1,y2表示按钮的左上角坐标为（x1,y1），右上角坐标为（x2,y2）,表示按钮的位置与大小，str表示按钮上输出的字符串，类型为LPCTSTR主要是为了使用easy.x中的输出文字函数outtextxy。之后的共有函数包括构造函数，以及按钮的绘制函数graph（）、判断按钮是否按下的函数buttondown（）与获得字符串的函数getstr（）。

之后的其他共有函数均为虚函数，是之后特殊继承类的新增私有成员的共有接口，比如getfunction\_type（）就是functionbutton新增的function\_type的私有成员变量的公共接口，之后在functionbutton的命名空间中该函数有返回值，其他的都是abort（）的退出程序操作。剩下的函数同理：getcom\_fun\_type()函数是为了判断这个完整类型函数是线性函数还是常函数；getnumber（）是为了获得常函数的私有变量number，也就是在合格常函数的函数值；getsequence（）函数是为了获得非完整函数的函数序号，我将我设计的几个非完整函数进行了编号，比如sin函数序号为1，cos函数序号为2。这些虚函数的设计就能很好的实现多态，使得我对统一的button\*类型变量进行判断，这也为我之后的表达式解析做了很多铺垫。

### function类

function类的设计是为了构造一个对应关系，其实这里面最关键的刻画性质就是一个gety（）函数，但是对于不同种类的函数，这个gety（）函数的刻画方式也不一样。

#### 1.2.1、继承关系图C:\Users\DX\AppData\Local\Temp\WeChat Files\656b46a5d0718f556068e1ed454b600.png

#### 1.2.2、成员变量与成员函数说明

针对function抽象类，成员只有一个gety（double x）函数，但是gety（double x）函数为纯虚函数，必须等待之后的继承类对这个函数进行多态定义：

首先若函数为一个非组合起来的函数，将之分为两类，一类是完整函数，既不需要与不可能与另外一个函数进行复合，即线性函数与常函数，这两类的函数的gety（double x）函数可以直接定义，分别返回自变量本身或者常函数代表的常量；另外一类是非完整函数，这类函数均需要一个函数与之进行复合，当然这个函数可以是常函数或者一次函数，之后的gety（double x）就是对这个函数在x的返回值进行相关操作，有根据函数本身不一样，又继承出sin\_function、cos\_function等一系列初等函数，这些函数的gety（double x）操作也都分别调用cmath库当中的相应函数进行定义，最终达到一个对function\*->gety(double x)的多态编译。

同时我也定义了组合而来的函数类，也就是add\_function , minus\_function ， divide\_function，multipy\_function四类。这些类都需要用两个function指针来构造，然后他们的gety（double x）就对这个funtion指针的返回值做加减乘除操作，这样就定义好了所有情况的function，避免出现一个无法操作的函数。

## 算法设计与技术难点

### 2.1、表达式解析的算法设计

针对表达式解析这个最关键的问题，我将这个过程的核心部分抽象为一个函数function\* express\_button(vector<button\*> some\_bu),这个函数会将一个按钮数组进行解析，最后输出这个按钮数组表示的数学函数的指针。以下是这个过程具体的实现过程。

首先第一步通过检测是否有over键判作为循环的判断依据，一直对检测到的按钮用其指针进行数组的存储，最终得到一个vector生成的动态按钮指针数组。之后进行express\_buttton的调用。

Express\_button这个函数分为几个大的步骤，首先是检测获得第一个完整的函数，之后判断是否后续还有数据需要解析，如果需要，紧接着的按钮一定代表加减乘除某个运算，再对这个运算进行信息储存，然后之后紧跟四则运算的一个按钮指针作为起点，再次读取一个完整函数，并生成这两个函数组合的新函数，之后重复上述过程的后半部分，直到对这个数组解析完毕。对于获得完整函数的这个步骤，我又将它单独抽象出来定义一个function\* getcom\_fun ( vector<button\*>some\_bu ,int &begin\_read) 函数:

而get\_com\_fun这个函数的基本算法是：先检测阅读起点的按钮类型，如果为线性函数，就直接返回线性函数对象x的地址，如果为常函数按钮，就是一个数字，这里要考虑这个数字可能不止一位，我这里用了一个单独的函数getnumber来获得这个数字，之后获得这个number之后，用这个number构造一个常函数对象，将之储存到我预先定义好的对象数组中，返回地址即可。如果阅读起点为非完整函数，我就对之后的括号进行检测，从而获得紧跟着的这个括号里面的所有按钮指针，重新构成一个数组，调用自身这个express\_button函数对这个数组进行解析，之后用这个解析函数返回的函数指针构造相应的非完整函数对象，储存在预先数组中，并返回地址。如果是直接检测到括号的话，就对之后括号里的新数组进行解析即可。

以上就是表达式解析的核心过程的基本逻辑

### 2.2、技术难点及其解决方案

#### 2.2.1、关于如何检测over键并暂定button数据的读取

对于鼠标信息的读取以及判断按下了哪个按钮这个过程的循环的判断依据我定义了一个全局变量，但不是用static定义的静态全局变量，因为我想要在overbutton类的成员函数中对这个进行赋值，所以我使用了extern关键词来定义这个判断依据变量int ifover，最终很好的解决了这个问题

#### 2.2.2、如何对不合法输入的数据进行报错

对于这个问题，我使用了多态的知识来解决这个问题。对于button基类，我设计了很多虚函数来获得这个button具体的信息，但是有很多信息是只针对某一个继承类才有意义的，所以我就在其他的继承类或者基类当中定义这个虚函数的执行语句为abort（），也就是如果不是我想要的合法种类的button访问这个函数的话就会直接退出程序，就避免的更加复杂的switch之后报错的语句出现。当然这里面还有一些问题，就是我在检测number有几位的时候，必定会使得检测到不是常函数按钮，所以这个时候如果调用它的getcom\_fun函数就会出现abort（），对于这些函数我在基类就定义了某个代表no的返回值，而没有使用abort（），从而解决了这个问题。

#### 2.2.3、如何局部生成某个对象，但是可以返回一个具有全局性的地址

这个问题困扰我很久，就是我的function都是需要检测信息之后在函数体内这个局部命名空间中定义，但是我返回的地址不能是指向这个局部变量的，不然在函数调用结束之后，这个局部变量的内存就被释放了，就找不到这个对象了。所以我尝试了定义static的静态全局变量，但是这样就只能初始化一次，不能重复调用这个函数。最后我采用了数组储存的办法：我首先在一开始就定义了相关的对象数组，之后在每一次调用的时候都将这个局部变量赋值到这个数组不同的数据中，控制这个数据序号的变量是一个检测解析函数共被调用几次的全局变量int several。这样就可以很好的解决这个问题进行操作。

#### 2.2..4、如果定义两个函数之间的加减乘除

对于这个问题，我也思考了很多可能的解决方案。一开始我想直接定义一个运算符来解决这个问题。但是我发现，我无法对一个对象中的成员函数进行篡改或者赋值，虽然这个对象最关键的刻画因素就是这个成员函数，但是每一个类的成员函数都是提前有所刻画的，不能任我摆布，而且我也无法再我的函数体内来定义一个新的函数gety（）。最后我设计了四个新的继承类，也就是add\_function , minus\_function ， divide\_function，multipy\_function四类，这四类的构造函数的形参都包含两个函数指针，同时他们的私有成员中也是这样的两个指针变量，这样我通过返回对私有成员的成员函数返回值加减乘除得到的数来定义它自身的gety，就很好的解决了一个问题，而避开了重载运算符这个麻烦的思路。

以上就是我的相关设计思路与一些问题的解决，虽然经历了很多磕绊，但是我对自己一步一步自己构建得到的逻辑链条还是非常满意的，希望之后可以对这个过程做进一步的改进！